# A content-aware bridging service for publish/subscribe environments

Jose M. Lopez-Vega[a,*], Javier Povedano-Molina[a], Gerardo Pardo-Castellote[b], Juan M. Lopez-Soler[c]

[a]*CITIC-UGR, Periodista Rafael Gomez 2, 18071 Granada, Spain*
[b]*Real-Time Innovations, 232 E. Java Drive, Sunnyvale, CA 94089 United States*
[c]*ETSIIT-UGR, Periodista Daniel Saucedo Aranda, 18071 Granada, Spain*

## Abstract

The OMG DDS (Data Distribution Service) standard specifies a middleware for distributing real-time data using a publish-subscribe data-centric approach. Until now, DDS systems have been restricted to a single and isolated DDS domain, normally deployed within a single multicast-enabled LAN. As systems grow larger, the need to interconnect different DDS domains arises. In this paper, we consider the problem of communicating disjoint data-spaces that may use different schemas to refer to similar information. In this regard, we propose a DDS interconnection service capable of bridging DDS domains as well as adapting between different data schemas. A key benefit of our approach is that is compliant with the latest OMG specifications, thus the proposed service does not require any modifications to DDS applications. The paper identifies the requirements for DDS data-spaces interconnection, presents an architecture that responds to those requirements, and concludes with experimental results gathered on our prototype implementation. We show that the impact of the service on the communications performance is well within the acceptable limits for most real-world uses of DDS (latency overhead is of the order of hundreds of microseconds). Reported results also indicate that our service interconnects remote data-spaces efficiently and reduces the network traffic almost N times, with N being the number of final

*Corresponding author. Tel.: 0034958241777. Fax: 0034958240831
*Email addresses:* `jmlvega@ugr.es` (Jose M. Lopez-Vega), `jpovedano@ugr.es` (Javier Povedano-Molina), `pardo@rti.com` (Gerardo Pardo-Castellote), `juanma@ugr.es` (Juan M. Lopez-Soler)

data subscribers.

*Keywords:* Bridging, Interoperability, Data Distribution Service, Middleware, Network communications

## 1. Introduction

Data Distribution Service (DDS) (OMG, 2006) is a specification adopted by the Object Management Group (OMG) aimed to standardize publish/subscribe communications in distributed scenarios.

In the recent years, DDS has been deployed in many contexts ranging from mission critical systems to financial environments. All these scenarios have in common that different collocated entities exchange data samples coming from different sources (for example, position sensors, stocks, etc.) in a well controlled environment (i.e., a data-space). In addition, this information should be exchanged ensuring certain guarantees such as reliability or deterministic time-delivery.

To deal with these requirements, DDS proposes a data-centric model (exchanged data is not opaque to the middleware) along with a rich set of quality of service (QoS) profiles that can be tuned to fit the communication demands of each specific scenario.

DDS was originally conceived for isolated local-area networks (LANs) in which data sources (publishers) and consumer entities (subscribers) are located in the same geographical location. Problems arise if a subscriber has interest in data being originated in a different data-space, especially if that data-space is separated by a Wide Area Network (WAN). This is the case of a company that relies on DDS for delivering some data-content internally, and needs to share a subset of that data-content with another company.

A simple solution to the previous use case is just to merge the two data-spaces. That is, publishing and announcing all the topics from one data-space to the other one. However, merging data-spaces directly creates some relevant problems.

The first problem is related to scalability. Every publication available in one data-space will be indiscriminately (and maybe uselessly) announced in the other data-space.

Other problems are concerned to data compatibility and confidentiality. For instance, each data-space may have its own (possibly different) data model (i.e., data names, types, and/or definitions), even if they refer to the same data item. For example, temperature data might be defined as Celsius in one data-space and Fahrenheit in the other one. Hence, for keeping unchanged the end-to-end application logic, data should be transparently transformed when bridging data-spaces with different data models. As an additional benefit, this will ease the integration between new and legacy DDS applications. Regarding confidentiality, some parts of the data should be confined in a data-space whereas other data should be public.

Finally, another relevant issue is related to the possibly different delivery requirements of the interconnected data-spaces. Namely, system administrators should be able to establish the QoS requirements that domains must met in order to be bridged.

To overcome these issues, in this paper we propose the ***DDS data-spaces Interconnection Service*** (referred to as DDS-IS), a service for enabling the seamless and efficient interoperation of entities located in two different (possibly remote) DDS data-spaces.

DDS-IS establishes a content-aware interconnection service between different data-spaces with the following distinctive features: a) **Scalability**. The overall traffic load is drastically reduced in comparison with direct data-space merging; b) **Data model compatibility and confidentiality**. The seamless communication between data-spaces with dissimilar data models (topics of different names or data-types) was not possible so far. DDS-IS solves this issue by properly transforming and/or isolating topic samples, if necessary. c) **Delivery guarantees**. DDS-IS establishes QoS requirements for bridged data-spaces; d) **Standard compliance**. The design of DDS-IS is fully compliant with the OMG DDS specification, hence it is implementation agnostic. e) **Performance**. Conducted experiments demonstrate that the impact of the proposed service on communications performance is well within the acceptable limits for most real-world uses of DDS.

The remainder of the paper is organized as follows. In Section 2 we provide a basic DDS background and discuss related work. In Section 3, we motivate our work and identify the service requirements. In Section 4, we describe the proposed DDS-IS architecture and the interactions among DDS-IS architectural elements. In Section 5, we include relevant implementation details. In Section 6, we address the conducted experiments and analyze the obtained results. Finally, in Section 7 we summarize the main conclusions of our work and discuss some open issues for future research.

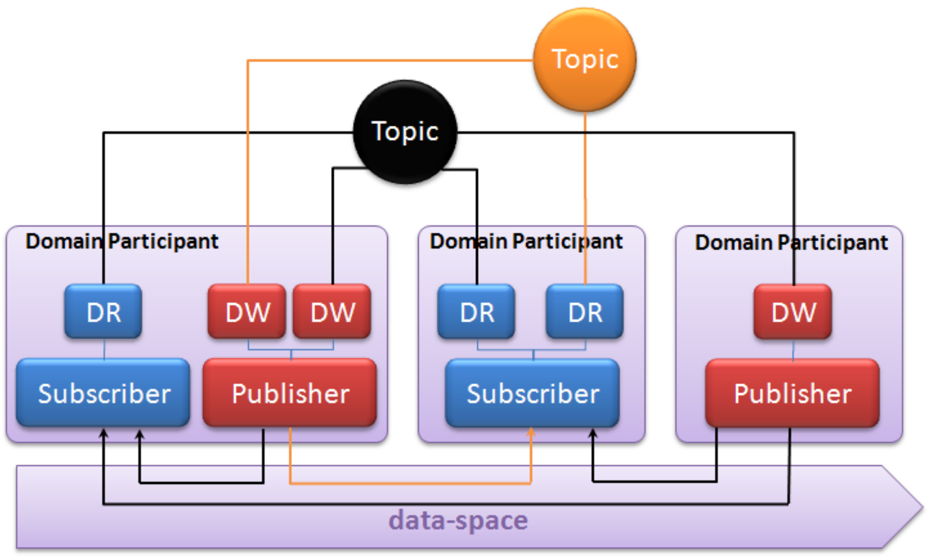


Figure 1: DDS concepts and entities.

## 2. DDS Background and Related Work

In this section, we overview the Data Distribution Service and we review some works related to the addressed problem.

### *2.1. Data Distribution Service Concepts and Features*

The DDS specification (OMG, 2006) defines the Data Centric Publish Subscribe (DCPS) programming model and an API for developing distributed applications.

According to the DCPS, publishers and subscribers exchange data items identified by a label (called Topic) within a shared global data-space (also referred to as Domain). This data-space is a distributed cache, whose content is accessed and updated using topic-specific endpoint entities, respectively called DataWriters (DW) and DataReaders (DR). Applications access to all these entities through DomainParticipants, which are the interface to DDS data-spaces.

Under the DCPS model, when a publisher wants to update data-content belonging to a certain topic, the middleware assumes responsibility for managing its delivery to the proper interested peers (i.e., to the associated subscribers). As a result, since DDS relieves the application from burden of transmitting and managing the data, the application logic is drastically simplified. Figure 1 illustrates main DDS concepts and relationships.

Data-Centric oriented Middleware contrasts with Message Oriented Middleware (MOM) in that message content is not opaque to the middleware. In this sense, DDS interprets the exchanged data-samples and is able to provide advanced functionalities such as data-caching or content-based-filtering.

DDS states that within a data-space data objects are completely identified by a topic name and optionally by a key. The key is useful to identify different instances of the same topic. Moreover each topic has a unique data-type or schema associated with it.

Each DW (or DR) has its own QoS policies and data-caching attributes. As an example, a DW and interested DRs can negotiate best effort or reliable topic delivery (RELIABILITY QoS). It is also possible to guarantee a maximum period between data updates (DEADLINE QoS), or to filter samples according to their content or timing. In case of QoS infringement, the middleware will detect it and optionally trigger actions.

DDS does not enforce any particular lower level preference or implementation requirement such as the use of a particular underlying transport protocol. For the sake of vendors interoperability, the OMG defined the so-called DDS Interoperability Wire Protocol, Real-Time Publish Subscribe (DDSI-RTPS) (OMG, 2008) protocol. The DDSI-RTPS specification determines the data representation, the message formats, and a mandatory Simple Discovery Protocol (SDP).

SDP allows DDS applications for automatic publication and subscription discovery. In this sense, SDP allows to build peer-to-peer architectures without the need of any server/broker. This functionality is achieved using a set of built-in publications (i.e., a set of mandatory-to-implement publications), which are handled using the so-called Built-in DataReaders (B-DR) and Built-in DataWriters (B-DW).

As mentioned before, the DDS specification states that a topic has an associated data-type. However, binding a data-type with a publication can cause problems for facilitating system evolution, for using a priori unknown data-types, for accomplishing backward compatibility, and for maintaining different data-types views.

The requirement of a single data-type has been relaxed in the in-progress DDS Extensible Types specification (DDS-XTYPES, (OMG, 2011)). However, the types are restricted to be compatible following the rules defined in the DDS-XTYPES specification. Namely, the specification defines a new set of topic types, known as *DDS Dynamic Topic Types*. As we will describe in Section 5.4, DDS-IS benefits from *DDS Dynamic Topic Types* for accessing arbitrary topic data-types.

## *2.2. Related Work*

In recent years publish/subscribe communication systems are gaining momentum and popularity. The publish/subscribe model allows publishers and subscribers to de-couple in time, space, and synchronization (Oh et al., 2010; Eugster et al., 2003).

Opposed to the request-response interaction model, MOM communication software is based on asynchronous message-passing. Java Messaging Service (JMS) (ORACLE, 2010) and Advanced Message Queue Protocol (AMQP) (AMQP-WG, 2011) are MOM-based technologies that allow to deliver messages between peers using a publish/subscribe approach. Other relevant approaches are also remarkable. For instance, Schmidt and O'Ryan (2003) propose a publisher/subscriber architecture for distributed real-time systems based on CORBA middleware, and Yoo et al. (2006) propose an XML-based solution for deploying publish/subscribe systems. More recently, in (Russello et al., 2011) the authors propose a publish/subscribe middleware for sense-and-react applications.

MOM technologies are usually implemented as brokered services, hence leading to well known issues derived from centralized architectures such as single point of failure and bottleneck. With these limitations in mind, the interconnection of different isolated domains may incur in an extra load to the brokers.

To cope with these issues, many solutions have been proposed in the literature. For instance, Carzaniga et al. (2001) propose SIENA, an event notification service that addresses the scalability problem using a publish/subscribe approach. The authors of SIENA study two different architectures for distributing event notifications: hierarchical client/server and peer-to-peer. According to their results, while the hierarchical approach is suitable for systems with low densities of clients that subscribe/unsubscribe very frequently, the peer-to-peer approach performs better when the total cost of communication is dominated by notifications.

The Apache QPiD project (Apache Foundation, 2011) includes a federation mechanism for AMQP that increases the scalability of the system by establishing dedicated routes between brokers. Marsh et al. (2010) provide an study of the design of efficient federation using QPiD. A different approach is included in (Yoo et al., 2009), where the authors combine clustering, content-based routing, and document flooding for providing scalable publish/subscribe communications for MANETs. In all these cases they deal only with scalability problem, leaving open issues such as data transformation.

Service federation and interoperability are issues that have also deserved consideration in grid environments. For instance, Fox et al. (2005) propose a hierarchical multi level (clusters and super-clusters) JMS compliant approach for interconnecting grids featuring scalability, load-balancing and fault-tolerant federation. More recently, in (Kertész and Kacsuk, 2010) the Grid Meta-Broker Service is proposed for interconnecting Grid islands.

Related to DDS, in (Corradi and Foschini, 2009; Corradi et al., 2010) REliable and VErsatile News delivery support for aGEncies (REVENGE) is presented. REVENGE is a middleware used to distribute breaking news among heterogeneous clients guaranteeing quality of service, availability, and reliability. REVENGE has two main contributions: a routing substrate to facilitate reliable data delivery between mobile nodes and a relay-based DDS support infrastructure with limited overhead to enable scalable, Internet-wide data dissemination. REVENGE relay nodes allow for the exchange of information among different DDS domains. However, this communication is bound to a specific topic/data-type, which limits the flexibility of the system.

Finally, in (Park et al., 2011) the authors propose a architecture to interconnect DDS data-spaces with Enterprise Service Bus (ESB), thus allowing to exchange information between the embedded world and the enterprise system. This work is mainly focused on increasing the interoperability of DDS by providing a routing substrate between ESB and DDS by proper data transformations. However, it does not provide any mechanism to increase the scalability on the DDS system.

The work presented here is complementary in the sense that it focuses in solving simultaneously three issues that currently exists in DDS: system scalability, data transformation between data-spaces, and QoS adaptation between remote entities. To the authors knowledge, it is the first work (compliant with OMG standard) that solves such issues by proposing a QoS-aware topic bridging architecture for content-aware data-spaces interconnection.

## 3. Motivations and Requirements

DDS is a middleware for solving data exchange on real-time distributed systems. In this sense, DDS-based solutions are usually implemented in industrial deployments. Two problems associated to these deployments are system heterogeneity and scalability. As systems grow in complexity, incompatibility problems arise among different software versions or among applications of different providers. In the same way, the natural evolution of enterprises generates scalability problems, and it is necessary to support bigger scenarios, perhaps even deployed among distant locations.

From the above problems, it emerges the need of a DDS bridging service that improves DDS scalability and eases the interoperation of heterogeneous DDS-based applications. Specifically, we have identified a set of requirements for developing this new service, which will be called the ***DDS data-spaces Interconnection Service*** (referred to as DDS-IS):

R1: *Domain bridging.* DDS-IS must connect different DDS data-spaces. These data-spaces are groups of publishers and subscribers that exchange data using compatible DDS middleware implementations and transport configuration.

R2: *Topic bridging.* DDS-IS must connect different DDS topics. A topic defines a data-type associated to a group of exchanged data-samples. Consequently, in order to bridge different topics our service must be able to adapt data-samples between different data-schemas.

R3: *Type-based transformations.* In order to adapt data-samples between different data-schemas, DDS-IS must be able to perform type-based data transformations. As an additional requirement, the service must allow users for defining their own transformations.

R4: *Content-based filtering.* DDS-IS must be able to use samples' content for determining if a sample should be bridged, or simply silently dropped.

R5: *QoS policies consistence.* DDS-IS must be compatible with existent DDS QoS profiles. DDS-IS must allow administrators for establishing what QoS requirements domains must met in order to be bridged. The service must ensure that the QoS policies of the interconnected entities are fully compatible with the ones configured for the DDS-IS. In addition, DDS-IS must allow administrators for integrating data-spaces with different (perhaps even incompatible) sets of QoS.

R6: *Transparency.* DDS-IS must not require modifications to existing DDS applications. This feature will enable legacy and new DDS applications to communicate through the proposed service. In this sense, our service will be an interoperability solution for connecting dissimilar data-spaces.

R7: *Standard compliance.* DDS-IS must be compliant with DDS standard family. This will ensure the interoperability of the service with any DDS-compliant middleware implementation.

R8: *Performance and scalability.* DDS-IS must have a low impact on DDS applications performance, mainly in terms of latency overhead. In addition, the system should improve DDS deployments scalability in terms of overall network traffic load.

R9: *Information confidentiality.* DDS-IS must provide mechanisms for filtering out sensitive information from bridged data-samples. Specifically, DDS-IS must be able to use data-samples' content for determining what data-samples should be discarded (this functionality is covered by R4). DDS-IS must also be able to selectively filter specific data-sample fields (instead of the whole sample) using the functionality covered by R3.

In the following section we describe the design of our system to satisfy the identified requirements.

## 4. System Design

To explain the proposed design, in this section we introduce a deployment example, we describe the DDS-IS architecture, and finally we study the interactions among DDS-IS components.

### *4.1. Example Scenario*

To ease the reading of the following sections, in Fig. 2 we include an example deployment scenario for the DDS-IS. Specifically, this example shows how DDS-IS can be used in an olive oil factory. Although other scenarios are possible, this example illustrates the functionalities of the proposed service.

The olive oil factory in our example generates data-content in two different environments, namely centrifugation system and storage system. In addition, there is a third environment (accounting and general supervision system) that stores the most relevant data-updates.

Decanter centrifugation is the part of olive oil production that separates olive oil from vegetation water and solid materials. In order to produce high quality oil, operators must control the centrifugation speed, centrifuge temperature, and oil density. The centrifugation system of our example has three centrifuges. Each one publishes viscosimetric information and temperature

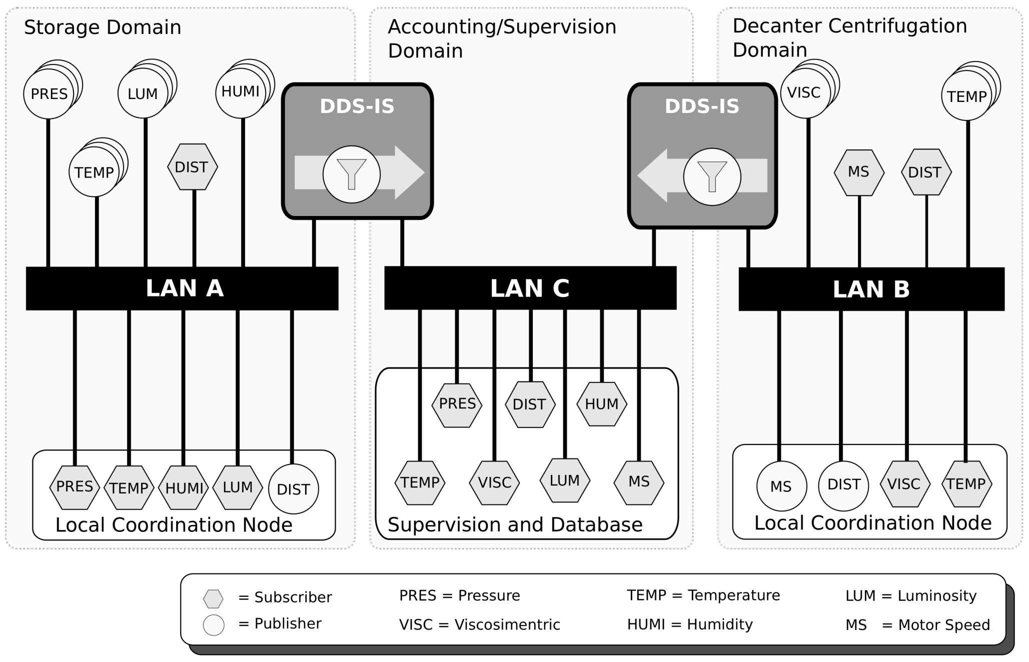


Figure 2: Example deployment scenario.

information (with a ratio of one update per second). Each centrifuge is also subscribed to two topics: one for controlling centrifugation speed, and other for controlling the input and output valves. A local coordination node controls the decanter centrifugation system, by subscribing to temperature and viscosimetric topics, and controlling centrifuge speed and valves.

Storage is the part of olive oil production where the oil is stored until its bottling and distribution. In order to conserve the quality of the oil, the system must monitor the pressure, temperature, humidity and luminosity of the storing tanks. In our example, we have assumed three oil tanks. Each one of these tanks publishes pressure, temperature, humidity and luminosity information (with a ratio of one update per second), and is subscribed to a valve control topic (i.e. three publishers per topic for a total of 14 publishers, and three subscribers). A local coordination node controls the valve status, and monitors published metrics (using four different subscribers and one publisher). Local coordination node can react to received values if necessary (for example, by activating a refrigeration system).

Although the generated data-content is mainly useful within its associated system (centrifugation or storage), the administrators of this factory want to store the most relevant events of each system in a central data-base, generating alerts if certain trigger values are reach. However, administrators do not want to modify the current sensor deployment, nor the local coordination nodes logic. Moreover, it is desirable to keep the data-content of each system isolated (in order to avoid overflowing local networks with unnecessary information).

In Fig. 2 two different DDS-IS have been deployed: one for bridging and filtering data-content from the storage system (i.e. from LAN A to LAN C), and the other for bridging and filtering data-content from the centrifugation system (i.e. from LAN B to LAN C). Using this deployment, DDS-IS nodes bridge only relevant data-samples to the accounting and general supervision system, while they avoid overflowing storage and centrifugation systems with unnecessary updates.

### *4.2. Architecture*

The DDS-IS architecture adopts a layered design based on the following motivations. First, it increases system modularity, by allowing the specific implementation of each layer to be changed without compromising system operation. Second, it allows existing applications to remain independent of new proposed DDS-IS entities while benefiting from the service. And third, it simplifies the overall understanding of the system, classifying entities of different functionality in different layers.

Fig. 3 depicts the architecture of our solution. It is composed of a set of components located on different nodes. The components of each node are arranged in four layers, namely, application, routing, pub/sub, and network layer. In the following subsections we explain the functionality of each layer and its contained components.

#### *4.2.1. Application Layer*

The application layer is the topmost layer of our architecture, and it is populated by DDS applications. These applications exchange information through DDS data-spaces by means of the underlying DDS middleware.

In our design, entities on this layer do not communicate directly with the routing layer. Instead, applications exchange data through the pub/sub layer using the existing standard DDS API (i.e., in the same way as if the DDS-IS was not present). This feature allows applications to be independent of the DDS-IS, and therefore they do not require any changes.

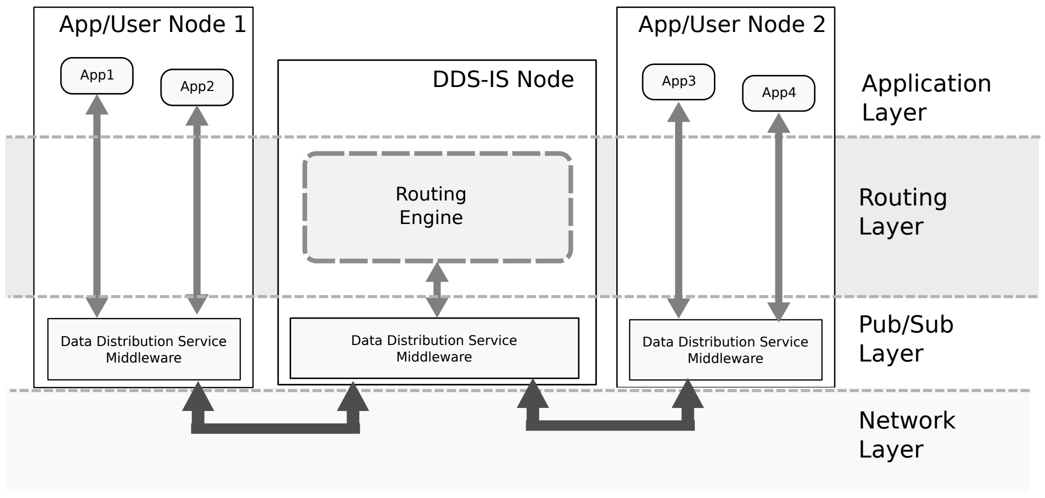


Figure 3: Layered DDS-IS architecture.

DDS-IS operation is totally transparent to the application layer. From the perspective of application layer entities, the only noticeable change is the increase in the data-content that they can access. This is because the DDS-IS enables the interoperation of applications associated with different data-spaces (i.e., different DDS domains), virtually combining the content of the involved data-spaces.

The DDS-IS also enables applications with different data models to exchange data-content. This is provided by the ability of the DDS-IS to transform data on the fly. Again, these transformations are independent of the application layer, and therefore already existing applications require no modifications for benefiting from the interconnection service.

As we have stated before, our solution does not modify the interactions between applications and DDS data-spaces. This is also valid for DDS compatibility checks. In this sense, if a DW (e.g. an application DW) and a DR (e.g. a DDS-IS DR) determine that they are not fully compatible (by performing the necessary DDS discovery-checks), they will not start exchanging data-content. Consequently, in order to bridge two applications with different data-types, or associated to different data-spaces, the DWs and DRs of the DDS-IS have to be properly configured.

In the same way, the QoS profiles associated to bridged entities must be compatible with the QoS profiles configured for the DDS-IS. As we will discuss later in Section 5.5, DDS-IS administrators can configure the service for enforcing the same QoS configuration for the two interconnected data-spaces. Alternatively, system administrators can configure DDS-IS for integrating two domains with different (perhaps even incompatible) sets of QoS policies.

*4.2.2. Routing Layer*

The routing layer is the core of our design. This new layer extends the functionality of the DDS specification: it allows for bridging DDS data samples between different DDS domains (i.e., between DDS data-spaces) and between different DDS topics, optionally transforming and filtering the exchanged data.

As a result, from the application layer perspective, two different (perhaps even remote) data-spaces appear as a single (and extended) DDS data-space. This functionality is provided by the Routing Engine (see Fig. 4), which contains the following components:

**Discovery Monitor**: This component processes the discovery events generated by the underlying DDS middleware (i.e., by B-DRs). Whenever a change occurs in the DDS discovery information, the DDS middleware notifies the Discovery Monitor. The Discovery Monitor then retrieves the received discovery information from B-DRs and delivers it to the proper Domain Route.

**Data Engine**: This component controls one or more DRs and DWs in the pub/sub layer. Data Engine has two missions. First, it takes received data samples from the pub/sub layer, and pushes those samples to the proper Topic Route. Second, it delivers resulting samples (from Topic Routes) to DWs for being published. There is one Data Engine associated to each Domain Route.

**Domain Route**: A mapping between two different DDS domains (i.e., data-spaces). Domain Route contains multiple Auto Topic Routes and Topic Routes. Domain Route interacts with the Discovery Monitor for obtaining discovery information from a set of B-DRs in the pub/sub layer. After receiving discovery updates from the Discovery Monitor, the Domain Route triggers the creation of DDS and DDS-IS entities according to the service configuration. A DDS-IS instance can contain multiple Domain Route instances, each one bridging a pair of DDS domains.

**Topic Route**: A mapping between an input topic (i.e., the topic from where the DDS-IS takes data) and an output topic (i.e., the topic where the DDS-IS publishes data). Therefore, a Topic Route is defined by an input

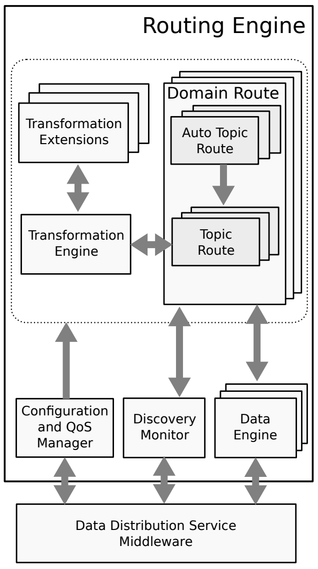


Figure 4: DDS-IS Routing Engine.

topic name and type, and an output topic name and type. Additionally, a Topic Route can apply transformations to data-samples using the Transformation Engine (explained below). The behavior of a Topic Route is as follows: first, it receives data from its associated Data Engine; then, it applies transformations (if any); and finally, it delivers processed data to the Data Engine for their publication. A Domain Route instance can contain multiple Topic Route instances, each one bridging a pair of DDS topics.

**Transformation Engine**: This element receives input data samples from a Topic Route, applies a set of transformations to those samples, and returns the result of the transformations. If a Topic Route requests for multiple transformations, the Transformation Engine processes them in sequence and following a predefined order (this is important because the output of a transformation and the input of the subsequent transformation must be type-compatible). Supported transformations are defined through multiple Transformation Extensions.

**Transformation Extension**: This element receives input data samples, applies a transformation to those samples, and returns the result of the transformation. To be able to process data of any type, Transformation Extensions use *DDS Dynamic Topic Types*. In order to increase system flexibility, A DDS-IS instance can contain multiple Transformation Extension instances, each one describing a particular data transformation. Transformation Engine can load external Transformation Extensions implemented by DDS-IS users.

**Auto Topic Route**: A generic Topic Route that bridges a range of topics. The topic name and type for bridged topics are not known *a priori*, and therefore Auto Topic Route provides auto-configurable topic bridging. Auto Topic Routes are characterized by a range of topic names or topic types for which the automatic setup is allowed. A Domain Route instance can contain multiple Auto Topic Route instances, each one covering a different set of topics.

**Configuration and QoS Manager**: Following the philosophy of the DDS standard, our design relies on a set of QoS policies for configuring the behavior of routing layer elements. Indeed, some of these QoS policies also control the configuration of entities belonging to the DDS core layer (for example, Topic Route QoS profiles also control the configuration of the associated DW and DR). The Configuration and QoS Manager is the component that controls the proper configuration of the DDS-IS.

### *4.2.3. Pub/Sub and Network layers*

The pub/sub layer (see Fig. 3) contains the entities associated to the DDS middleware (described in Section 2.1), and interacts with system's upper layers using the standard DDS API.

The pub/sub layer allows the DDS-IS to access to DDS data-spaces. In order to provide this functionality, the pub/sub layer perform the following tasks: (1) it notifies the routing layer about changes in discovery information, (2) it receives new data samples and delivers these samples to the routing layer, and (3) it publishes data obtained from the routing layer to DDS data-spaces.

In addition to the entities already introduced in Section 2.1, this layer includes two DDS WaitSets. A DDS WaitSet is an standard DDS entity that can be used to wait for specific DDS Events. DDS WaitSets are awaken whenever certain pre-defined trigger conditions are met. Specifically, our system uses the following two DDS WaitSets:

**Discovery WaitSet**: This element receives updates from B-DRs (see Section 2.1). It notifies the Discovery Monitor in the routing layer when a change occurs to discovery. This allows the DDS-IS to react to the arrival and departure of entities to/from the connected data-spaces.

**Data WaitSet**: This element notifies its associated Data Engine in the routing layer about the reception of new data samples in one of its DRs. This allows the DDS-IS to obtain DDS samples from the DDS core layer as soon as they are received.

The network layer is the lowest layer of the proposed architecture (see Fig. 3). It represents the platform-dependent components that are necessary for exchanging data-samples (such as the operating system, computer hardware, or communication network). One of the advantages of using a network middleware (such as DDS) is that it decouples the application logic from the particularities of the network layer. Consequently, we do not need to define any requirements for this layer, except that the chosen DDS implementation must be DDSI-RTPS-compliant.

### *4.3. System Operation*

In order to illustrate the interactions among the previously described entities, in this section we provide sequence diagrams for the two main system use cases.

#### *4.3.1. New DDS Entities Discovery*

DDS discovery notifies DDS-IS about the creation or deletion of DDS entities in any of the interconnected domains. This information enables DDS-IS to react to the changes in the input and output data-spaces.

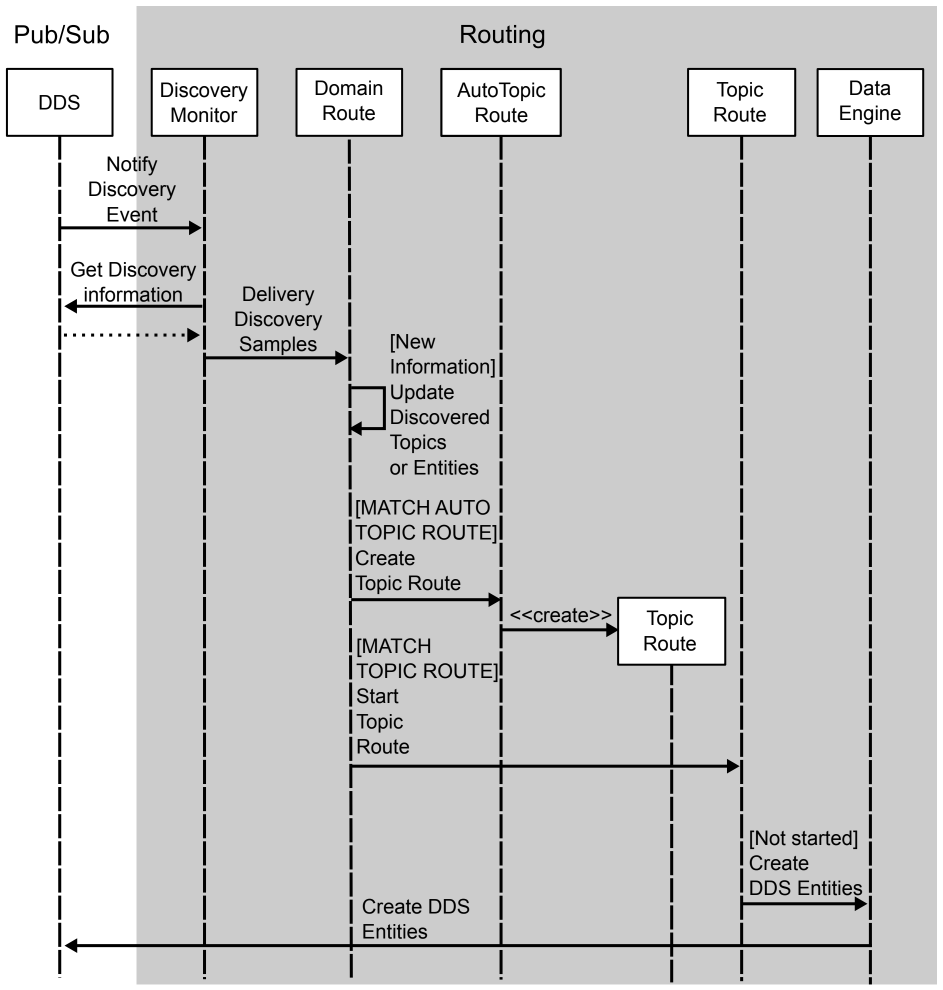
The sequence diagram of the discovery procedure is depicted in Fig. 5. Specifically, this sequence diagram describes how the discovery of DDS entities triggers the initialization of a Topic Route and all its associated entities.

A B-DR initiates the process when it receives a change in the discovery information of a DDS entity located in the same data-space (but associated to a different node). This event triggers a Discovery WaitSet (studied in Section 4.2.3), which then sends a notification to the Discovery Monitor.

Once the Discovery Monitor receives a discovery notification, it retrieves the kind of the originating B-DR (participant, publication or subscription), and the discovery information received by the B-DR.

At this point, the Discovery Monitor delivers all the received discovery samples to the Domain Route associated to the originating B-DR. Received samples can contain multiple changes in the discovery, which are processed separately. These changes can refer either to the creation or removal of remote DDS entities.

Using the received information, the Domain Route updates an internal register that stores information of previously discovered DDS entities and



Pub/Sub
Routing
DDS
Discovery Monitor
Domain Route
AutoTopic Route
Topic Route
Data Engine
Notify Discovery Event
Get Discovery information
Delivery Discovery Samples
[New Information] Update Discovered Topics or Entities
[MATCH AUTO TOPIC ROUTE] Create Topic Route
<<create>>
Topic Route
[MATCH TOPIC ROUTE] Start Topic Route
[Not started] Create DDS Entities
Create DDS Entities

Figure 5: Discovery sequence diagram.

topics. This register enables the DDS-IS for determining if it is necessary to create new Topic Routes upon the reception of discovery information.

Then the Domain Route checks if any of its active Auto Topic Routes matches with the received discovery information. If necessary, the Domain Route then consequently creates new Topic Routes according to Auto Topic Routes' configuration.

Finally, the Domain Route checks the existing Topic Routes, and then determines if the discovered DDS entity is associated to any of them. If the Domain Route finds a matching Topic Route, the Domain Route will check

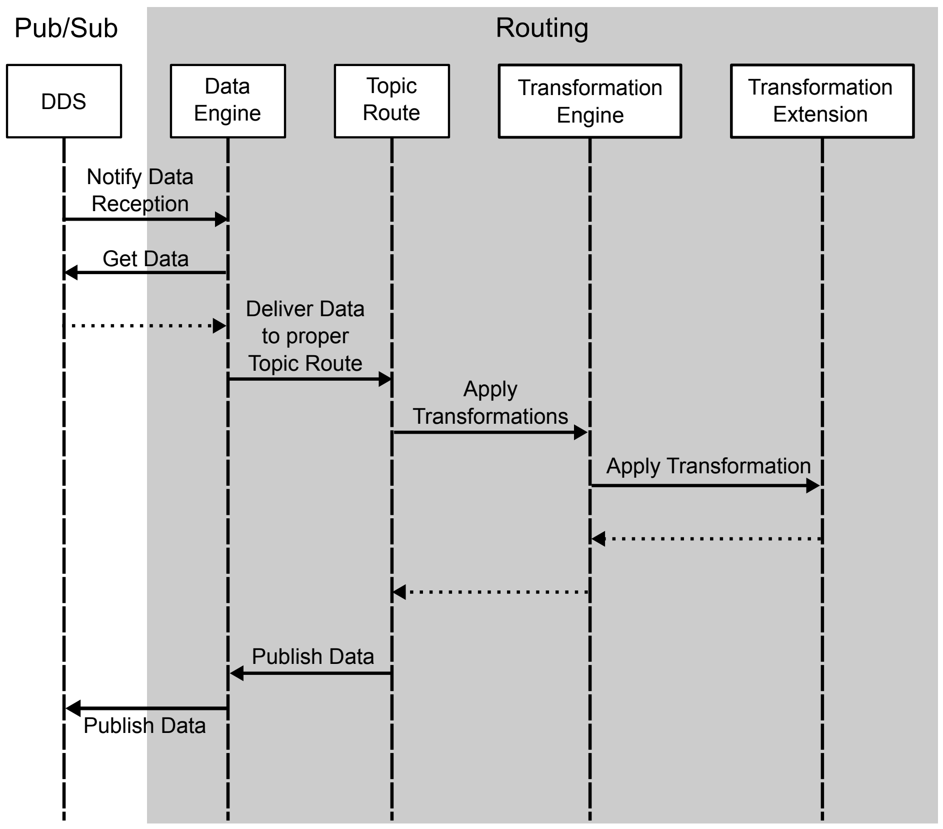


Figure 6: Data bridging sequence diagram.

if the Topic Route is already active. If not, the Domain Route starts the Topic Route (i.e., the Topic Route requests the Data Engine for creating the DR and the DW necessary for performing topic bridging).

### *4.3.2. Data Bridging*

Data Bridging is the process of communicating data-samples between two different DDS data-spaces. If the input and output data-spaces share the same topic, the DDS-IS can bridge data-samples without performing any modifications to data-samples' structure. However, the DDS-IS can also bridge data-samples between two data-spaces that do not share the same data-model. In this case, the DDS-IS must transform input data-samples for adapting them to the format required at the output data-space.

Fig. 6 depicts the sequence diagram of the data bridging process. A DR (located in the pub/sub layer) initiates the process when it receives a new data-sample from a DW located in the same data-space (but in a different node). This event triggers a Data WaitSet, also located in the pub/sub layer, which then sends a notification to the Data Engine associated to the DR.

After receiving the notification from DDS, the Data Engine retrieves all of the received samples from the originating DR, and delivers those samples to the Topic Route associated to that DR.

At this point, the Topic Route applies configured transformations (if any) to the input data-samples (using the Transformation Engine). Data Bridging then finishes with the publication of resulting data-samples to the output data-space by means of the proper Data Engine's DW (located in the pub/sub layer).

## 5. Implementation

After defining the DDS-IS architecture and its design, we have implemented a proof-of-concept prototype. This prototype was used to validate the approach, perform field-testing, and to evaluate the performance impact of our service (for further information regarding conducted experiments and obtained results, please refer to Section 6). In this section, we explain the most relevant implementation decisions that we have taken during the prototype development.

### *5.1. DDS Middleware Implementation*

The DDS-IS prototype was implemented using RTI's Data Distribution Service version 4.4d (RTI, 2012). We have chosen this implementation because it supports the *DDS Dynamic Topic Types* extension. This extension is a new API for the DDS standard family that is being standardized by the OMG (OMG, 2011).

Nevertheless, our design is implementation-independent, as it only uses standardized DDS features plus the *DDS Dynamic Topic Types* extension. Consequently, the proposed design can be implemented using any other DDS-compliant implementation (such as OpenSplice DDS (PrismTech, 2012)).

Indeed, the implemented prototype is also compliant with the DDS Interoperability Wire Protocol (DDSI-RTPS) (OMG, 2008). Consequently, DDS-IS can interconnect any DDSI-RTPS-compliant data-space regardless of its particular implementation.

Regarding the used programming language and bindings, we have implemented our prototype in C, and we have compiled the source code using gcc version 4.4.3.

## *5.2. Discovery Propagation*

As we have mentioned, a Topic Route connects an input data-space's topic (i.e., a set of application DWs) with an output data-space's topic (i.e., a set of application DRs). To enable the interoperation of these DWs and DRs, the DDS-IS must create both a DR at the input domain and a DW at the output domain.

In this scenario, the question naturally arises: when should the service create the input/output DR/DW? The answer to this question depends on the particular application requirements. If system resources (memory and CPU) are scarce, DDS-IS should not create the paired DW-DR until a match between the input topic and the output topic is found (i.e., both the matched DR and DW have been discovered). However, if we need all of the discovery information to propagate through the DDS-IS, the service should create both the DR and DW as soon as a matched publication or subscription is discovered. Alternatively, it may be preferable for the system to set up Topic Route's entities as soon as the user configures a Topic Route (even if the DDS-IS does not find any matching DWs or DRs).

In order to support scenarios with different requirements and constraints, we decided that the prototype would allow the user for configuring the triggering conditions for the creation of Topic Route's entities. In Section 5.6, we provide an example of how to configure the creation for the input and output of a Topic Route.

## *5.3. Propagating DataWriter Information*

By default, DDS middleware generates an unique ID (using hosting machine information and implementation specific parameters) for each DW. In the early stages of DDS-IS prototype implementation, this was also the behavior of the service. However, after some testing we concluded that this behavior prevents application DRs from distinguishing between original data-samples and replicas of those data-samples.

The ability of distinguishing between original data-samples and their replicas is specially critical for deployments with redundant DDS-IS nodes. In these scenarios, this functionality may significantly impact system resources consumption, given that it avoids the delivery of duplicated samples to final applications.

Consequently, we decide to enable the DDS-IS to maintain the original DW information, which is contained in input data-samples. The implemented prototype allows configuring if the original DW information is maintained, or if it is replaced by the DDS-IS DW information. In this way, the DDS-IS administrator is able to configure the system behavior according to the specific scenario requirements.

### *5.4. Dynamic Types*

The use of static topic types is simple, efficient, and provides compile-time type-safety. However it has some limitations as it requires types to be known *a priori* and compiled into the code, something that is not possible in generic bridge services, such as the proposed DDS-IS.

In DDS, Type schemas (the names and definitions of a type and its fields) are represented by TypeObject objects. A type object value consists of a type object kind and a list of members.

In addition to using serialized type objects locally, the DDS-XTYPES specification (OMG, 2011) states that a serialized form of these (called the TypeRepresentation) must also be published automatically during DDS discovery.

This mechanism allows applications to discover the types, and use the *DDS Dynamic Topic Types* API (also defined in the DDS-XTYPES specification) to publish or subscribe to topics of types unknown at the time of compiling.

As previously mentioned, this is a key feature for providing the DDS-IS. This is because it enables the DDS-IS to interact with any DDS applications, even if those DDS applications' data-schemas are not known at compilation time.

### *5.5. Quality of Service*

As we described in Section 4.2.2, DDS-IS relies on DDS QoS profiles for configuring the behavior of the service. Consequently, DDS-IS also relies on DDS built-in compatibility-checking mechanisms for ensuring that QoS profiles of communicating entities are fully compatible.

DDS-IS QoS checking is performed on a hop-to-hop basis (i.e., between DDS-IS entities and application entities located within the same domain). This feature allows system administrators for decoupling QoS for different domains, enabling the integration of heterogeneous (and perhaps even incompatible) scenarios. Of course, administrators can configure the same QoS policies for the two interconnected domains, which guarantees that QoS remain invariant across interconnected data-spaces.

Regarding QoS enforcement, DDS QoS policies were initially intended for working within single domains. Consequently, QoS enforcement is applicable only among entities within a particular domain (i.e., hop-to-hop) for most of DDS QoS policies. There are, however, two exceptions: LIVELINESS and LIFESPAN. These two DDS QoS policies can be enforced on a end-to-end basis (i.e., across multiple interconnected domains).

End-to-end enforcement for the rest of DDS QoS policies is out of the scope of this paper, as it would require modifying the behavior of the DDS middleware itself. Besides, it would break backwards compatibility, which is stated in requirement R6.

For an analysis of the impact of DDS-IS on each DDS QoS policy please refer to Appendix A.

## *5.6. XML Configuration Format and XSD Schemas*

The DDS-IS uses XML files to load its configuration. XML (Extensible Markup Language) is a set of rules for encoding documents. These rules are defined in the XML 1.0 Specification produced by the W3C (W3C, 2008).

Fig. 7 provides an example of a basic XML configuration file; specifically, it includes the following XML tags:

**dds**: The root of the XML document; it can also include the definition of QoS policies for the system.

**dds_is**: This tag includes the configuration of a DDS-IS instance. Therefore, every other DDS-IS should be included as a child of the routing_service tag.

**domain_route**: This contains the configuration for a Domain Route and must contain a participant_1, a participant_2, and a session tag.

**participant_1** and **participant_2**: These tags define the input and output DDS domain for the service using the domain_id tag.

**session**: This contains the configuration for a set of Topic Routes and Auto Topic Routes that share the same Data Engine instance. Session tag can contain multiple topic_route and auto_topic_route tags.

**auto_topic_route** and **topic_route**: The system uses these tags to set up the Auto Topic Routes and Topic Routes.

```xml
<?xml version="1.0"?>
<dds xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../schema/dds_is.xsd">
  <dds_is>
    <domain_route name="DefaultDomainRoute" enabled="true">
      <participant_1>
        <domain_id>0</domain_id>
      </participant_1>
      <participant_2>
        <domain_id>1</domain_id>
      </participant_2>
      <session name="DefaultSession" enabled="true">
        <auto_topic_route name="All" enabled="true">
          <publish_with_original_info>
            true
          </publish_with_original_info>
          <publish_with_original_timestamp>
            true
          </publish_with_original_timestamp>
          <input participant="1">
            <creation_mode>IMMEDIATE</creation_mode>
          </input>
          <output>
            <creation_mode>IMMEDIATE</creation_mode>
          </output>
        </auto_topic_route>
      </session>
    </domain_route>
  </dds_is>
</dds>
```

Figure 7: XML configuration file example.

**publish_with_original_info**: This determines if the DDS-IS maintains the original DW information for publishing data to the output.

**publish_with_original_timestamp**: This determines if the DDS-IS maintains the original timestamp information for bridged data-samples.

**input**: This defines the configuration of the route input. It allows configuring filters for topic routing, the QoS policies for the input DR, and the creation_mode, which is defined below.

**output**: This defines the configuration of the route output. It allows configuring filters for topic discovery at the output, the QoS policies for the output DW, and the creation_mode, which is defined below.

**creation_mode**: This configures the way discovery information is propagated from the input to the output (or *vice versa*). In this way, when a DR is discovered at the output, the system can set up the corresponding Topic Route immediately (as in the example) or wait until a DW is discovered at the input.

## 6. Validation and Performance Evaluation

To study the operation and performance of the DDS-IS prototype, we conducted a set of experiments in a controlled LAN environment. Specifically, we measured the impact of the DDS-IS in terms of round-trip latency, throughput, and network traffic load.

### *6.1. Experimental Set-up*

The test-bed was composed of six Core i5 at 2.40GHz-2.66GHz machines (named lab01, lab02, lab03, lab04, lab05, and lab06) running Linux Kernel 2.6.32-22 x86_64 (Ubuntu 10.04) and the *RTI DDS 4.4d* middleware. These machines communicate through a 24-port Gigabit switch with VLAN support. In our experiments, these nodes receive one of three possible roles:

**Source-node**: A node that generates DDS samples, receives responses for those samples, and calculates experimental statistics.

**Echo-node**: A node that receives DDS samples from source-nodes and generates responses.

**Routing-node**: A node that bridges two different networks.

We developed a benchmarking tool for conducting our experiments. This benchmarking tool consists of two components:

**PerformanceTest_tester**: This application runs in the source-nodes. It publishes samples into a topic called `Ping` and receives them from a subscribed topic, called `Pong`. `Ping` samples contain a sequence number, the publication timestamp, and a variable size payload as well. The structure complexity of `Ping` payload is variable. `Pong` samples only contain a sequence number and the original `Ping` publication timestamp. This timestamp is compared with the sample reception time for obtaining samples round trip time. Each `Ping` and `Pong` sample is sent in a separate message.

**PerformanceTest_remote**: This application runs in echo-nodes. It takes samples from the `Ping` topic and republishes their sequence number and timestamp field content into the `Pong` topic.

In the conducted tests, published samples performed a round trip between source-nodes and echo-nodes. This approach allows for the precise measurement of the sending and reception times without performing any clock synchronization. Consequently, at least two different routes have to be configured in the DDS-IS: the outgoing route (for bridging `Ping` samples) and the incoming one (for bridging `Pong` samples).

For all the reported evaluations, our benchmarking tool uses RTPS over UDP protocol, and sends each sample separately (i.e., each UDP datagram contains only one RTPS sample).

Regarding sample sizes, `Ping` samples contain a payload whose size ranges from 256 to 8192 bytes and a protocol overhead of 118 bytes. `Pong` samples have the same protocol overhead as `Ping` samples, but their payload has a size of 12 bytes because they only contain a sequence number and a timestamp.

We considered the following metrics for the DDS-IS performance evaluation:

**Throughput**: The average number of samples per time interval (seconds) received from the data-space by subscribers.

**Round-trip Latency**: The average end-to-end elapsed time for delivering samples from the original source-node to the echo-node plus the elapsed time for delivering samples back to the original source-node. Our tool measures the latency at application level. In this sense, publishers measure the time before delivering data-samples to DDS middleware, while subscribers measure the time after DDS delivers data-samples to the benchmarking tool.

**Generated traffic load**: The total generated traffic (in bytes) for a specific network segment in a particular scenario. This includes the payload plus all the overheads generated by the whole protocol stack.

In order to evaluate the proposed service performance, we conduct all the experiments in two different scenarios. Namely,

**No-DDS-IS Scenario**: Publishers and subscribers associated to source-nodes and echo-nodes were located in the same data-space, but in different networks. DDS entities communicate through one or more Layer-3 Linux Routers.

**DDS-IS Scenario**: Publishers and subscribers associated to source-nodes were located in a different data-space (also in a different network) than the ones associated to echo-nodes. DDS-IS interconnects the involved data-spaces.

In both scenarios, samples were published using the DDS Reliability QoS policy set to `RELIABLE`. Additionally, we have configured the QoS policy `KEEP_HISTORY` to `ALL_HISTORY`. This configuration will force the DDS middleware to ensure the delivery of all the samples. However, it can cause that few samples have an extremely high latency, potentially biasing the average results. In this respect, the reported results only include the 95th percentile latency.

We have set DDS DataReader heartbeats low-period (i.e. the one used when DataReaders' cache has less than 5 samples) to 100 milliseconds, and we have set DDS DataReader heartbeats high-period (i.e. the one used when DataReaders' cache has more than 15 samples) to 10 milliseconds.

We have also configured the DDS middleware for discovering DDS entities only in the interfaces of interest for each particular experiment. In this respect, all the experiments use a secondary network for launching the benchmarking tools without interfering on the tests.

As we explained in Section 5, the DDS-IS prototype has been implemented using RTI's C API. On the other hand, the benchmarking tool has been implemented using C++.

Regarding Linux kernel configuration parameters, we have used the default values associated to Ubuntu 10.04 for all the machines.

### *6.2. DDS-IS Impact in N to N Communications*

In this section, we evaluate the impact of the DDS-IS in terms of round-trip latency and throughput for different payload (sample sizes), data types, and DDS-IS configurations when N publishers and N subscribers communicate following a 1 to 1 relationship.

To make the results comparable, both no-DDS-IS and DDS-IS scenarios used the same network topology: two LAN networks (A and B) connected through a computer (lab02) with two Ethernet interfaces. In the no-DDS-IS

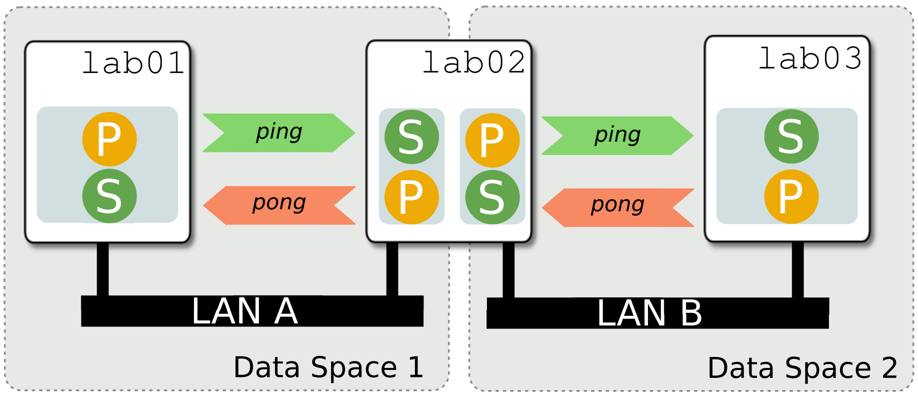


Figure 8: DDS-IS scenario for 1 to 1 performance experiments. In the N to N case, lab01 and lab03 run multiple publishers and subscribers.

scenario, lab02 is a single layer-3 Linux Kernel Router (we use just one data-space), whereas in the DDS-IS scenario, lab02 interconnects two different data-spaces and provides the proposed DDS-IS (see Fig. 8).

Each test consisted of publishing 500,000 samples per topic and using a specific payload size. These samples were published at source-node (lab01) into the `Ping` topic; after being received by `Ping` subscriber at echo-node (lab03), they were published back using the `Pong` topic.

To avoid transitory behavior (warm-up) effects, before starting the tests we force some samples to be exchanged after the DDS entities discovery.

### *6.2.1. Performance Impact in Simple 1 to 1 Communications*

This first set of experiments aims to evaluate the performance impact for different data types and DDS-IS configurations when only one topic for `Ping` and one for `Pong` is active. Although the `Pong` topic was always the same, this experiment uses the following `Ping` topic types:

**SimpleType (ST) 512, 1024 bytes**: These contain a sequence of random bytes, a sequence number, and a timestamp.

**ComplexType (CT) 512 bytes**: This structure contains two levels of nesting. It includes a SimpleType (256 bytes) data, four 64byte-string fields (two of them in a nested struct), a long field, and eight boolean fields.

**ComplexType (CT) 1024 bytes**: This structure contains three levels of nesting. It includes two ComplexType-512 byte structs.

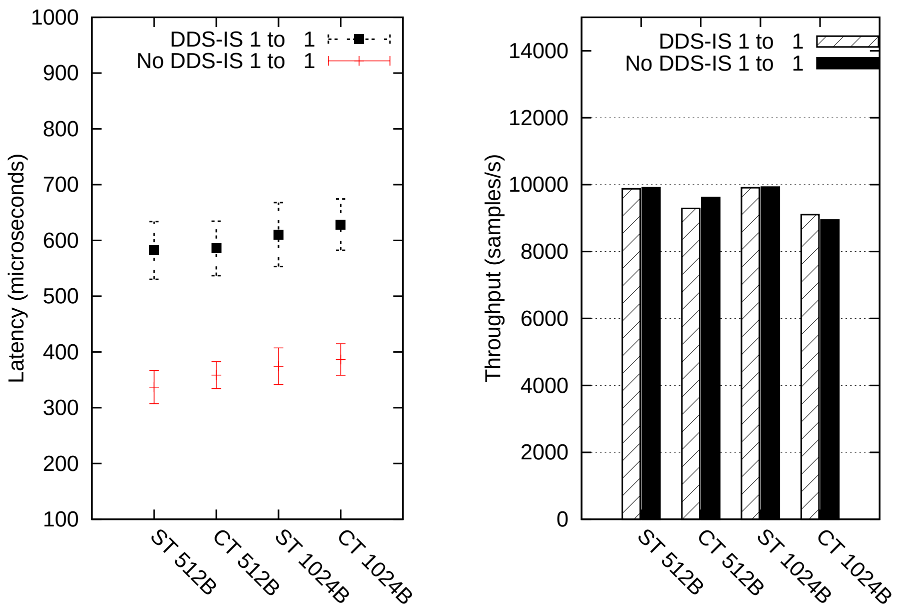


Figure 9: DDS-IS impact for different data types in 1 to 1 communications.

Fig. 9 shows the performance results of both no-DDS-IS and DDS-IS scenarios. In this experiment neither transformations nor filtering are enabled.

The results show that although the DDS-IS introduces approximately a round-trip latency overhead of 250 microseconds, it has no a significant impact on throughput.

We can also conclude that in this case data complexity does not have a significant performance impact on DDS-IS scenario in comparison to no-DDS-IS scenario. Consequently, in the remainder experiments we focus our study on testing different sample sizes.

### *6.2.2. Performance Impact of Data Transformation and Filtering Features*

The DDS-IS allows loading external data transformations and applying these transformations to the output data. In this sense, the performance degradation is highly dependent of the transformation complexity. Therefore, rather than reporting an exhaustive analysis, this section aims to evaluate the cost of using simple data transformations, which will show the overhead introduced by inclusion of an additional function call to the datapath.

Fig. 10 shows the incurred latency and throughput obtained for a simple data transformation definition (labeled as `CT` *`size`* `transf.`). In particular,

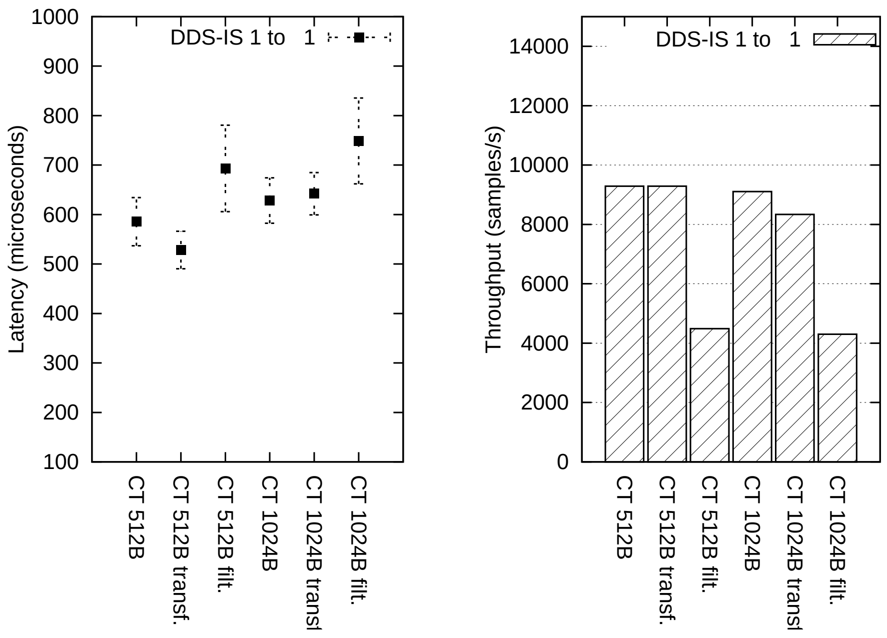


Figure 10: DDS-IS performance for filtering and transforming data.

we have set up the DDS-IS for changing a long field and a boolean field to a specific configured value. In this experiment, we observe that the complexity of the transformed structure degrades the throughput (see the obtained results for labels CT 512B transf. and CT 1024B transf.).

However, simple transformations have no significant negative impact on latency. Indeed, for CT512 samples, using transformations reduces the average latency. This behavior is because the additional processing time introduced by transformations reduces the number of context switches in the routing-node CPU, what reduces the average latency overhead.

Regarding data filtering test, we configured a filter that drops samples according to the content of the samples. Published samples are labeled with a number randomly obtained from a uniform distribution ranging from 0 to 999. The DDS-IS filters the samples according to the following definitions. For CT512 case (labeled in Fig. 10 as `CT 512B filt.`) the filter was:

```
(long_cond_00 < 500)
or
(nested_ping.sequence_number = 4294967295)
```

Whereas for CT1024 case (labeled in Fig. 10 as `CT 1024B filt.`) it was:

```
(nested_complexping_02.long_cond_00 < 500)
or
(nested_complexping_01.nested_ping.sequence_number = 4294967295)
```

The first condition on each filter drops approximately the 50% of the samples. The second condition ensures the warm-up signaling samples traverse the DDS-IS filter. According to the obtained results (see Fig. 10), the filter introduces an average overhead of 110 microseconds when it uses a filtering condition on the first level of the type CT512, and 140 microseconds for evaluating the condition in a second level of nesting of the CT1024. In addition, DDS-IS approximately halves the throughput compared to a scheme without filtering, as expected.

This experiment validates the filtering and data transformation features of the DDS-IS. We can conclude that for the evaluated conditions DDS-IS accomplishes these tasks without a significant degradation of system performance.

#### *6.2.3. Performance Impact in N to N Communications with Multiple Topics*

In this section, we study how DDS-IS affects performance as the number of concurrent distinct topics (and Topic Routes) increases.

In this case, we use the same topology of Fig. 8 but now we increase the number of bridged topics (up to 16). We always use an even number because our benchmarking tool utilizes two types of topics: `Ping` for outgoing samples and `Pong` for incoming ones.

This experiment is a worst-case scenario for the DDS-IS operation, as the capabilities of data multiplexing of DDS-IS are not used. Indeed, the DDS-IS has to maintain a pair of DR/DW per route. More particularly, the last case studied in this section creates 16 DWs and 16 DRs in the same DDS-IS. Although this is not an appropriate scenario of DDS-IS operation (multiple 1-1 routes), these experiments provide some insight on the DDS-IS limits.

Fig. 11 depicts the average round-trip latency as a function of the payload size. The results show that the DDS-IS performs reasonably well for samples with a size up to 6144 bytes when 2 to 16 concurrent routes (topics) are active. Specifically, the DDS-IS introduces an overhead from 250 to 400 microseconds for the round-trip latency.

For the 8192-sized payload, the scenarios with 8 and 16 concurrent topics reveal a performance degradation. In these cases the round-trip latency

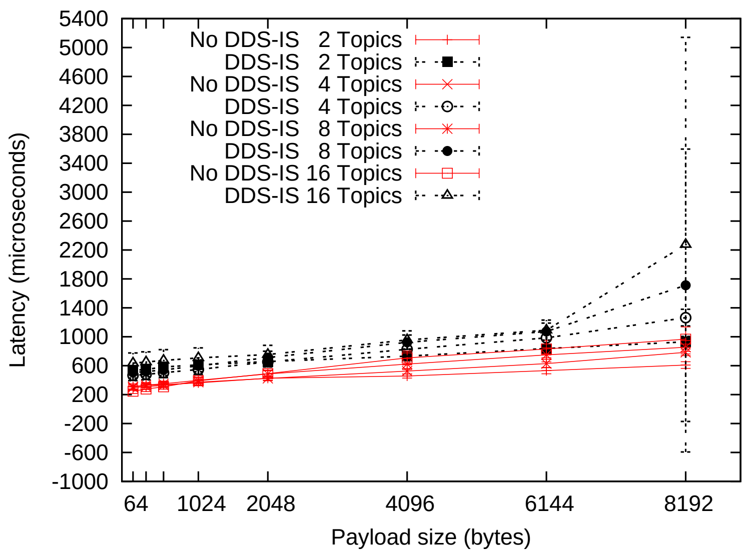


Figure 11: Average latency for different payloads and number of routes (topics) active.

overhead introduced by the DDS-IS increases to 450 microseconds and 1.3 ms respectively. Additionally, the standard deviation for round-trip latency reaches 2 ms, which indicates the DDS-IS performance degrades for this load level.

Regarding throughput, the results in Fig. 12 show that DDS-IS has an insignificant impact on the throughput for the scenarios with 2-4 concurrent topics.

For the 8 and 16 topics cases, the DDS-IS throughput achieves at most 30,000 samples per second (for the 64-byte payload), whereas the scenario without DDS-IS reaches 50,000 samples per second in the 8-topic case, and almost 75,000 samples per second in the 16-topic case (both for the 64-byte payload).

It is important to remark that in the 16-topic case the DDS-IS is publishing 30,000 samples in both directions (`Ping` samples from source-nodes to echo-nodes and `Pong` samples from echo-nodes to source-nodes). This is especially relevant in the 64-byte scenario, in which the size of `Pong` samples is similar to the size of `Ping` samples. In this scenario, while source-nodes and echo-nodes just manage one DW (for publishing `Ping` samples or for publishing `Pong` samples) and one DR (for receiving `Pong` samples or for receiving `Ping` samples), the routing-node has to manage a pair of DW/DR

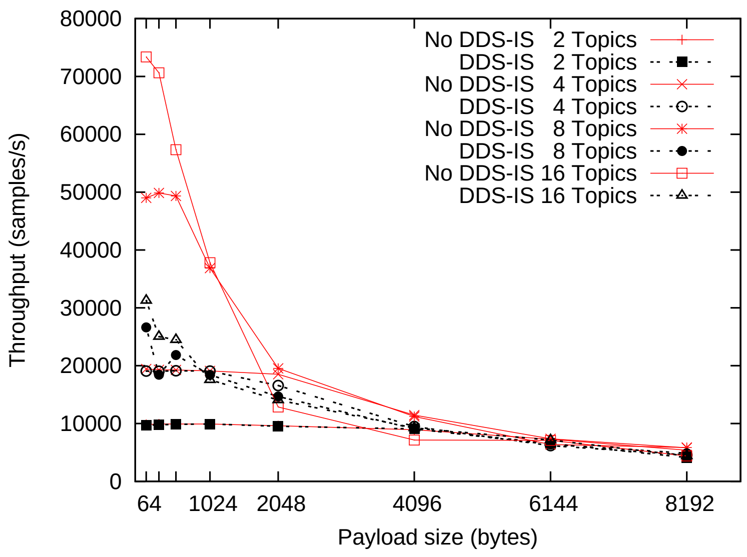


Figure 12: Total average throughput for different payloads and number of routes (topics) active.

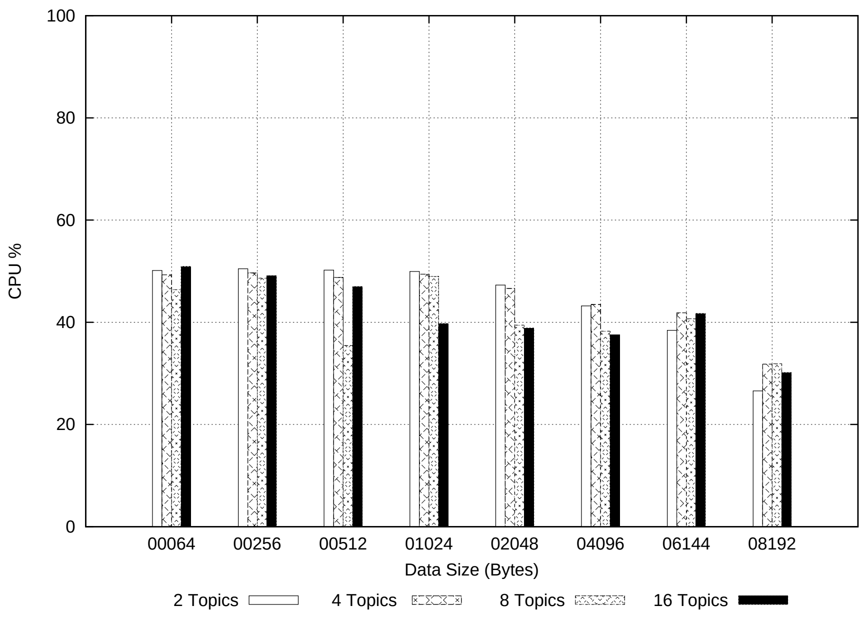


Figure 13: CPU Impact for different payloads and number of routes (topics) active.

per topic.

We have also measured the impact on CPU consumption for the routing-node (i.e., for lab02). Fig. 13 shows that the maximum CPU consumption is obtained for the experiments using small packets, which are the ones with a highest rate of samples per second. The obtained results also show that the degradation in performance for bigger sizes is not due to CPU saturation. Instead, it is mainly due to synchronization problems between the middleware and the used ethernet interfaces, along with the increased cost of packet disassembling, assembling, and loss-recovery.

### *6.3. DDS-IS Impact in M to N Communications*

In the previous section, we studied the limits of the DDS-IS in a scenario where no data multiplexing is present. The following experiments analyze the performance of the DDS-IS when several copies of certain published data are delivered to multiple subscribers.

#### *6.3.1. DDS-IS Performance Impact*

Fig. 14 shows the topology used in this set of experiments. Specifically, we run a `Ping` topic publisher and a `Pong` topic subscriber on node lab01, a DDS-IS in node lab02 (for the no-DDS-IS scenario, IP forwarding was enabled at this node), and 0-4 subscribers in each of the nodes lab03, lab04, lab05, lab06. To avoid overloading the DDS-IS with `Pong` samples, only one `Pong` publisher at node lab03 publishes `Pong` samples back to lab01 (for measuring average round-trip latency and throughput on lab01 node).

Fig. 15 depicts average round-trip latency results obtained for this set of experiments. The obtained maximum overhead is 330 microseconds (for the 8192-bytes payload, 1 to 1 experiment). For the 4096-bytes, 1 to 4 case, the DDS-IS scenario obtains almost zero round-trip latency overhead (852 and 846 microseconds for the cases with and without DDS-IS respectively).

Indeed, for a number of subscribers greater than 4, and a payload greater than 4096 bytes, we have obtained lower round-trip latencies values for the DDS-IS scenario than for the no-DDS-IS one. This is because the latency overhead introduced by the DDS-IS logic is offset by the reduction of the load on lab01 (the original publisher), the reduction of traffic in LAN A (this will be studied in next section), and the fact that it is more efficient to re-send lost samples from lab02 (in the same LAN) than from lab01 (located at two hops).

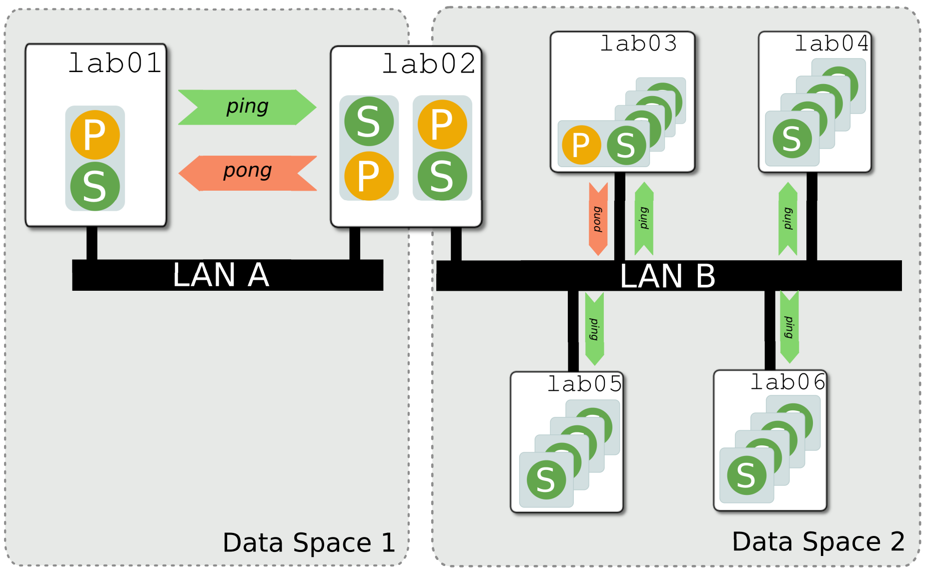


Figure 14: System topology used in 1 to N performance experiments.

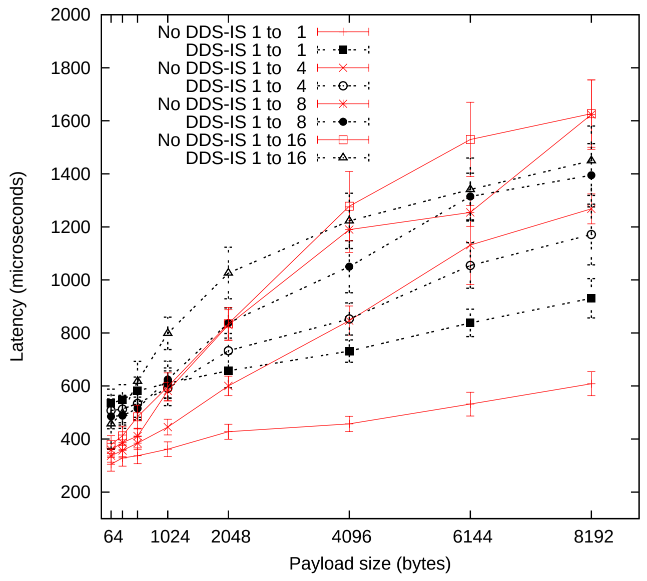


Figure 15: Average latency for different payloads when demultiplexing data from 1 to N.

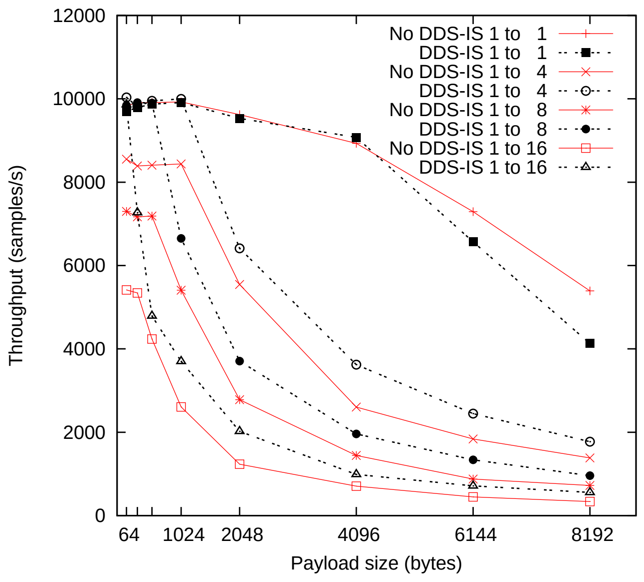


Figure 16: Throughput per subscriber for different payloads when demultiplexing data from 1 to N.

Regarding throughput, the results in Fig. 16 show that the DDS-IS scenario beats the no-DDS-IS scenario for all the experiments with a number of subscribers greater or equal than 4. This is especially relevant for the 64-byte, 1 to 16 scenario, where the DDS-IS maintains a rate of 10,000 samples per second per subscriber, while in the no-DDS-IS case the rate drops to 5400 samples per subscriber. The explanation for this behavior is the same we stated for latency experiment: network usage reduction on LAN A, reduction of load for original source-node, and the fact that re-sending from lab02 is cheaper than re-sending from lab01.

### *6.3.2. DDS-IS Scalability Impact*

Along this paper we stated that the DDS-IS can help in reducing the network traffic load. The following experiment (Fig. 17) will show our claim. The experimental set-up consists of 1-4 `Ping` publishers (running on lab01), 1-15 `Ping` subscribers (uniformly distributed among lab04, lab05, and lab06), and two DDS-IS (running on lab02 and lab03). To avoid overloading the DDS-IS with `Pong` samples, only one `Pong` publisher at node lab04 publishes `Pong` samples back to lab01. Although many other alternative settings can be envisaged, this simple scenario is useful for our goal. The configuration

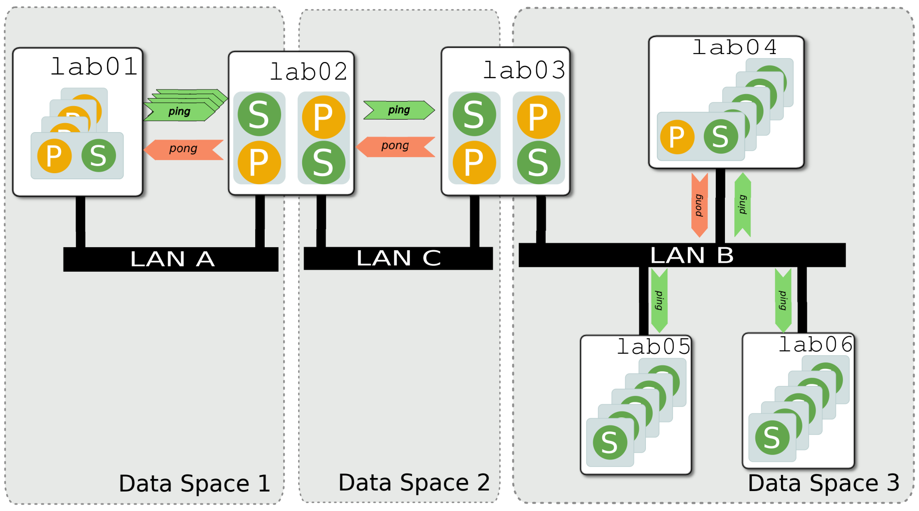


Figure 17: System topology used in M to N scalability experiments.

for the no-DDS-IS scenario is almost the same, although lab02 and lab03 just forward the samples between LAN A and LAN B.

The scenario without DDS-IS exchanges all the samples through a unique DDS domain (this case corresponds to just one data-space with no interconnection service), whereas the DDS-IS scenario presents three different data-spaces: one between lab01 and lab02; another one between lab02 and lab03; and another one among lab03, lab04, lab05, and lab06.

This configuration allows checking the potential of the DDS-IS, its ability to multiplex and demultiplex DDS traffic. In this respect, with this experiment we evidence that DDS-IS eases the integration of DDS systems across Wide Area Networks (WANs). Particularly, we point out that DDS-IS can isolate individual applications from WAN transport. For example, local applications can communicate using IP multicast for efficient data distribution while DDS-IS bridges remote sites using WAN infrastructures (maybe using reliable TCP connections).

The potential of the DDS-IS for reducing the network traffic can be explained using a simplistic example: assuming M publishers and N subscribers, a network without loss, and S number of samples published by each publisher, the number of total samples to the LAN C would be equal to MxNxS in the scenario without DDS-IS. On the other hand, if we use the configuration shown in Fig. 17, the traffic in LAN C decreases to MxS, given that only

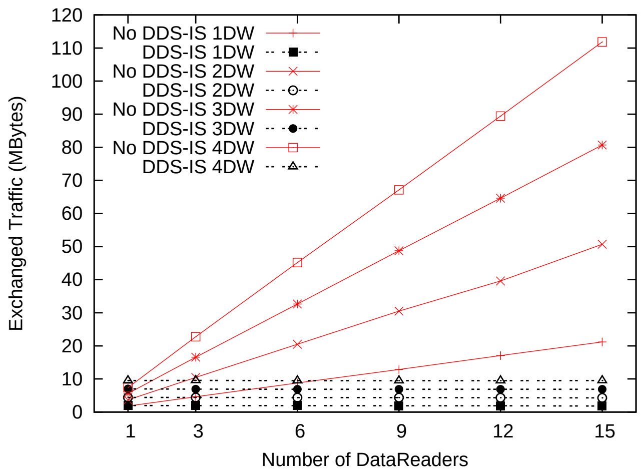


Figure 18: DDS traffic load on LAN C for both no-DDS-IS and DDS-IS scenarios.

one copy of each published sample is sent.

Additionally, according to the OMG standard DDS entities (publishers and subscribers) exchange discovery traffic. Whereas in the no-DDS-IS scenario this is a MxN problem, the DDS-IS only creates a DW on lab03 and a DR on lab02, what evidence that DDS-IS can reduce the complexity in LAN C to a much more simple discovery problem (1x1 for the particular considered setting).

Fig. 18 shows the total generated DDS traffic load on LAN C for the scenarios with and without DDS-IS. As expected, the obtained results show the traffic in the no-DDS-IS scenario increases almost linearly with the number of subscribers, whereas the traffic in DDS-IS scenario is constant for a growing number of subscribers. Thus, finally we conclude that the DDS-IS can help in saving network resources.

## 7. Discussion and Conclusions

In this paper, we propose the DDS data-spaces Interconnection Service (DDS-IS), a solution for one key identified limitation in DDS deployments: the interconnection of DDS data-spaces.

In addition to the interconnection problem, the DDS-IS also addresses three unresolved issues of DDS: data transformation between data-spaces, system scalability, and QoS adaptation between remote entities. Moreover, and following the data-centric philosophy of DDS, the proposed service is able perform content-based filtering of the exchanged data.

The design of the DDS-IS features an efficient data transforming capability, which transforms data as it flows between applications. As a result, multiple versions of the same application, or even applications of different vendors, can now interoperate, without change, despite differences in data structures and interface definitions. DDS-IS also supports content-based filtering, which can be used for avoiding sensitive information of a given data-space to be exposed to other data-spaces.

One of the key benefits of our solution is its standard compliance. The DDS-IS is fully compliant with the latest specifications of the DDS standard family. As a result, our design is not implementation-dependent, and hence it is applicable to any existing or future DDS implementation. Namely, the DDS-IS benefits from the DDS-XTYPES, a new OMG specification that is in the latest stages of its acceptance, for accessing to topic types discovered at execution time.

We have implemented a DDS-IS prototype that demonstrates the advantages of our proposal. This prototype is compliant with the DDS Interoperability Wire Protocol (DDSI-RTPS) (OMG, 2008). Consequently, DDS-IS can interconnect any DDSI-RTPS-compliant data-space regardless of its particular implementation.

Based on this prototype, we report experimental results to evaluate the performance impact of the proposed service. The reported results show that the DDS-IS has a very low impact on DDS performance, in terms of overhead latency and achievable throughput. Finally, we have demonstrated that the DDS-IS can improve the scalability of DDS systems by reducing the network traffic load between remotely interconnected data-spaces.

As future work, we are interested on researching in new possible extensions and applications of the proposed service. For instance, we are interested on studying how the Bloom-based SDP proposed in (Sanchez-Monedero et al., 2011) impacts on the scalability of DDS-IS-based deployments. Moreover, we hold that the DDS-IS can ease the integration of DDS systems across Wide Area Networks (WANs). However, for this goal some open issues need to be solved, as dealing with NAT transversal as well as defining new components to provide secure DDS communications.

We also want to study how DDS-IS can help in DDS domain federation. New signaling and coordination functions among DDS-IS instances could be designed for improving the overall system (i.e., to achieve load balance, robustness, and end-to-end QoS enforcement).

Cloud computing architectures have gained recently more and more attention in both industry and academia. In this regard, we would like to use DDS-IS for connecting DDS-based applications in Cloud scenarios. In particular, once NAT traversal and DDS federation issues were solved, DDS robustness and DDS-IS flexibility motivate further research in this area.

Finally, other general question that deserves attention is how DDS and future extensions (including the proposed DDS-IS) may help as an enabler technology in a future data-centric Internet.

## 8. Acknowledgment

We would like to thank Fernando Crespo-Sanchez and Alejandro De Campos for their support with the development of the prototype. This work has been partially supported by the Spanish Ministry of Science and Innovation (project TIN2010 20323).

## Appendix A. DDS-IS Impact on DDS QoS policies

Table A.1 studies the level of enforcement for each QoS policy: local (if a particular QoS is enforced locally by DDS middleware), hop-to-hop (if a particular QoS is enforced within a particular DDS domain), or end-to-end (if a particular QoS is enforced across interconnected domains). The table also contains a description of each QoS policy, and its corresponding compatibility checks. For the sake of simplicity, we have assumed that publisher=DataWriter and subscriber=DataReader.

Table A.1: Study of the level impact of DDS-IS on DDS QoS policies.

| **Name** | **Description** | **DDS standard compatibility requirements (checked hop-to-hop)** | **Level of enforcement** |
|---|---|---|---|
| USER_DATA | Attaches additional information to the created Entities. | None | N/A |
| TOPIC_DATA | Attaches additional information to the created Topics. | None | N/A |
| GROUP_DATA | Attaches additional information to the created publishers and subscribers. | None | N/A |
| DURABILITY | Controls whether DDS will make data available to late-joining readers. If this QoS is set to PERSISTENT, publishers send a set of past values (the actual number of samples depends on HISTORY QoS) to late joining subscribers. | Offered kind >= requested kind (PERSISTENT > TRANSIENT > TRANSIENT_LOCAL > VOLATILE) | Hop-to-hop<br>Historic samples are sent between publishers and subscribers within the same domain.<br>Although samples published within a domain will be normally bridged by the DDS-IS, there is not mechanisms for ensuring that those samples arrive to their final destination (see RELIABILITY QoS), even if both domains have DURABILITY set to PERSISTENT. |
| DURABILITY_SERVICE | Configures both HISTORY and RESOURCE_LIMITS | None | N/A |
| PRESENTATION | Defines the dependence across updates | Offered access_scope >= requested access_scope (GROUP > TOPIC > INSTANCE) | Local<br>Each individual subscriber ensures that samples are delivered to the corresponding application according to this QoS. |
| DEADLINE | Maximum time between two updates | Offered deadline_period <= requested deadline_period | Local<br>According to the standard, each individual publisher and subscriber enforces this QoS. In this sense, if DEADLINE period is missed, the particular publisher or subscriber generates a notification to its associated application. |
| LATENCY_BUDGET | Provides a hint about the priority of data-communications | Offered duration <= requested duration | N/A (hint)<br>According to the standard, this policy is considered a hint. There is no specified mechanism as to how the service should take advantage of this hint. |
| OWNERSHIP | Controls whether DDS allows multiple publishers to update the same instance | Offered ownership == requested ownership (SHARED or EXCLUSIVE) | Hop-to-hop |
| OWNERSHIP_STRENGTH | Determines the ownership of a data-instance. | None | Local<br>Each individual subscriber delivers to the application only the samples generated by the publisher with a higher OWNERSHIP_STRENGTH. |
| LIVELINESS | Controls the mechanism and parameters used by the Service to ensure that particular entities on the network are still alive. | Offered kind >= requested kind (MANUAL_BY_TOPIC > MANUAL_BY_PARTICIPANT > AUTOMATIC)<br>Offered lease_duration <= requested lease_duration | Hop-to-hop/End-to-end<br>DDS-IS supports the propagation of DDS entities creation and disposal across the bridged domains.<br>In addition, the DDS-IS could be configured for maintaining DDS-IS publishers and subscribers independently of the liveliness state of the entities within bridged domains. |
| | | | Continuing next page |

continued from previous page

| **Name** | **Description** | **DDS standard compatibility requirements (checked hop-to-hop)** | **Level of enforcement** |
|---|---|---|---|
| TIME_BASED-_FILTER | Indicates the minimum separation between two updates delivered by DDS to the application. | deadline period >= minimum_separation | Local<br>Each individual subscriber ensures that samples are delivered to application according to this QoS. |
| PARTITION | Subdivides a domain into subdomains. | Publishers and subscribers must use the same partition for communicating. | Hop-to-Hop<br>DDS-IS entities and DDS application entities only communicate if they are associated to the same PARTITION. |
| RELIABILITY | Ensures published samples are delivered to subscribers. | Offered kind >= requested kind (RELIABLE > BEST_EFFORT) | Hop-to-hop<br>Although DDS will ensure that samples published in a particular domain arrive to all the subscribers within that domain, there is not mechanisms for ensuring end-to-end enforcement of this policy.<br>In this sense, if a DDS-IS along the datapath crashes, the original publisher has not means for checking that samples have been received across bridged domains.<br>Fortunately, this problem can be mitigated using redundant DDS-IS nodes. |
| TRANSPORT-_PRIORITY | Allows the application to take advantage of transports capable of sending messages with different priorities. | None | N/A (hint)<br>According to the standard, this policy is considered a hint. There is no specified mechanism as to how the service should take advantage of this hint. |
| LIFESPAN | Limits the validity of each individual update (comparing samples timestamp with reception date). | None | Hop-to-hop/End-to-end<br>The level of enforcement of this QoS depends on the configuration of the publish_with_ original_timestamp DDS-IS parameter. publish_with_ original_timestamp controls whether samples maintain the original timestamp or if the timestamp is regenerated with each hop. |
| DESTINATION_ORDER | Controls how each subscriber resolves the final value of a data instance that is written by multiple publishers. | Offered kind >= requested kind (BY_SOURCE-_TIMESTAMP > BY_RECEPTION-_TIMESTAMP) | Local<br>Each individual subscriber will ensure samples are delivered to application according to this QoS. |
| HISTORY | Controls the number of historic samples stored by publishers. | None | N/A<br>Each individual publisher will store the a number of historic samples equal to the value of this policy. |
| RESOURCE-_LIMITS | Controls the resources consumed by DDS. | None | N/A<br>Each individual publisher or subscriber will limit the maximum allocated resources according to this policy. |
| ENTITY_FACTORY | Controls the behavior of a DDS Entity as a factory for other DDS entities. | None | N/A |
| WRITER_DATA_LIFECYCLE | Controls the behavior of the publisher with regards to the lifecycle of the data-instances it manages | None | N/A |
| READER_DATA_LIFECYCLE | Controls the behavior of the subscriber with regards to the lifecycle of the data-instances it manages | None | N/A |

## References

AMQP-WG, 2011. Advanced Message Queuing Protocol (AMQP).
URL `http://www.amqp.org/`

Apache Foundation, 2011. Apache QPid.
URL `http://qpid.apache.org/`

Carzaniga, A., Rosenblum, D. S., Wolf, A. L., Aug. 2001. Design and evaluation of a wide-area event notification service. ACM Trans. Comput. Syst. 19 (3), 332–383.
URL `http://dx.doi.org/10.1145/380749.380767`

Corradi, A., Foschini, L., May 2009. A DDS-compliant P2P infrastructure for reliable and QoS-enabled data dissemination. In: Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on. Vol. 0. IEEE, Los Alamitos, CA, USA, pp. 1–8.
URL `http://dx.doi.org/10.1109/IPDPS.2009.5160957`

Corradi, A., Foschini, L., Nardelli, L., 2010. A DDS-compliant infrastructure for fault-tolerant and scalable data dissemination. In: Computers and Communications (ISCC), 2010 IEEE Symposium on. pp. 489–495.
URL `http://dx.doi.org/10.1109/ISCC.2010.5546756`

Eugster, P. T., Felber, P. A., Guerraoui, R., Kermarrec, A. M., Jun. 2003. The many faces of publish/subscribe. ACM Comput. Surv. 35 (2), 114–131.
URL `http://dx.doi.org/10.1145/857076.857078`

Fox, G., Lim, S., Pallickara, S., Pierce, M., 2005. Message-Based Cellular Peer-to-Peer Grids: Foundations for Secure Federation and Autonomic Services. Future Generation Computer Systems 21 (3), 401–415.
URL `http://dx.doi.org/10.1016/j.future.2004.04.010`

Kertész, A., Kacsuk, P., Apr. 2010. GMBS: A new middleware service for making grids interoperable. Future Generation Computer Systems 26 (4), 542–553.
URL `http://dx.doi.org/10.1016/j.future.2009.10.007`

Marsh, G., Sampat, A. P., Potluri, S., Panda, D. K., 2010. Scaling Advanced Message Queuing Protocol (AMQP) Architecture with Broker Federation and InfiniBand. Tech. rep.
URL `ftp://ftp.cse.ohio-state.edu/pub/tech-report/2009/TR17.pdf`

Oh, S., Kim, J.-H., Fox, G., Mar. 2010. Real-time performance analysis for publish/subscribe systems. Future Generation Computer Systems 26 (3),

318–323.
URL http://dx.doi.org/10.1016/j.future.2009.09.001

OMG, 2006. Data-Distribution Service for Real-Time Systems (DDS).v1.2. Tech. rep., OMG.
URL http://www.omg.org/cgi-bin/doc?formal/07-01-01.pdf

OMG, 2008. Real-time Publish-Subscribe (RTPS) Wire Protocol DDS Interoperability Wire Protocol Specification.v2.1. Tech. rep., OMG.
URL http://www.omg.org/cgi-bin/doc?formal/09-01-05.pdf

OMG, 2011. Extensible And Dynamic Topic Types For DDS (DDS-XTypes) 1.0 (FTF 2) - Beta 2. Tech. rep.
URL http://www.omg.org/spec/DDS-XTypes/1.0/Beta2/

ORACLE, Oct. 2010. Java Message Service (JMS).
URL http://www.oracle.com/technetwork/java/index-jsp-142945.html

Park, Y., Chung, D., Min, D., Choi, E., 2011. Middleware Integration of DDS and ESB for Interconnection between Real-Time Embedded and Enterprise Systems. Convergence and Hybrid Information Technology, 337–344.
URL http://www.springerlink.com/index/U5P0U30775807686.pdf

PrismTech, 2012. OpenSplice DDS Data Distribution Service.
URL http://www.prismtech.com/opensplice

RTI, 2012. Real-Time Innovations (RTI) DDS Data Distribution Service.
URL http://www.rti.com/

Russello, G., Mostarda, L., Dulay, N., Apr. 2011. A policy-based publish/subscribe middleware for sense-and-react applications. Journal of Systems and Software 84 (4), 638–654.
URL http://dx.doi.org/10.1016/j.jss.2010.10.023

Sanchez-Monedero, J., Povedano-Molina, J., Lopez-Vega, J. M., Lopez-Soler, J. M., May 2011. Bloom filter based discovery protocol for DDS middleware. Journal of Parallel and Distributed Computing 71 (10), 1305–1317.
URL http://dx.doi.org/10.1016/j.jpdc.2011.05.001

Schmidt, D. C., O'Ryan, C., Jun. 2003. Patterns and performance of distributed real-time and embedded publisher/subscriber architectures. Journal of Systems and Software 66 (3), 213–223.
URL `http://dx.doi.org/10.1016/S0164-1212(02)00078-X`

W3C, Nov. 2008. Extensible Markup Language (XML) 1.0 (Fifth Edition). Tech. rep., W3C.
URL `http://www.w3.org/TR/REC-xml/`

Yoo, S., Son, J. H., Kim, M. H., Dec. 2006. An efficient subscription routing algorithm for scalable XML-based publish/subscribe systems. Journal of Systems and Software 79 (12), 1767–1781.
URL `http://dx.doi.org/10.1016/j.jss.2006.03.039`

Yoo, S., Son, J. H., Kim, M. H., Jul. 2009. A scalable publish/subscribe system for large mobile ad hoc networks. Journal of Systems and Software 82 (7), 1152–1162.
URL `http://dx.doi.org/10.1016/j.jss.2009.02.020`

## Appendix B. Vitae

**Jose M. Lopez-Vega** is a Ph.D. student at the Department of Signal Theory, Telematics and Communications of the University of Granada (Spain). He received his B.S. degree in Telecommunications from the University of Granada, and his M.S. in Multimedia Systems also from the University of Granada. He was granted a Ph.D. fellowship by the Education Spanish Ministry on July 2009 and started his Ph.D. studies. His research interests include QoS, peer-to-peer networks, mobile networks, and network middleware.

**Javier Povedano-Molina** is Ph.D. candidate Department of Signal Theory, Telematics and Communications of the University of Granada (Spain). He holds a M.Sc. degree in Computer Science from the University of Granada and a B.S. in Computer Science from the University of Cordoba (Spain). His research interests include multimedia networking, peer-to-peer architectures, distributed systems, and Cloud Computing.

**Gerardo Pardo-Castellote** is Chief Technology Officer of Real-Time Innovations, Inc. He holds a Ph.D. in Electrical Engineering, an M.S. in Computer Science, and an MSEE from Stanford University (USA). He also holds a B.S. in Physics from the University of Granada (Spain). His professional experience includes time- critical software for data acquisition and processing, real-time planning and scheduling software, control system software, and software-system design. He authored the original version of RTI Data Distribution Service. Gerardo currently chairs the Data Distribution Service Finalization Task Force at the Object Management Group (OMG).

**Juan M. Lopez-Soler** is Associate Professor at the Department of Signals, Telematics and Communications of the University of Granada (Spain). He teaches Computer Networks, Data Transmission, and Multimedia Networking courses. He holds a B.Sc. and a Ph.D. in Physics from the University of Granada. In 1991-92, he joined the ISR at the University of Maryland (USA) as Visiting Faculty Research Assistant. He has participated in 10 public and 13 private research projects, 9 as coordinator. He has published 13 papers in indexed journals and more than 25 in international workshops/conferences. His research interests include real-time middleware and multimedia networking.