Evaluating Technologies for Tactical Information Management in Net-Centric Systems

Ming Xiong, Jeff Parsons, James Edmondson, Hieu Nguyen, and Douglas Schmidt
Vanderbilt University, 2015 Terrace PL, Nashville, TN, USA 615-343-8197

ABSTRACT[†]

Recent trends in distributed real-time and embedded (DRE) systems motivate the development of tactical information management capabilities that ensure the right information is delivered to the right place at the right time to satisfy quality of service (QoS) requirements in heterogeneous environments. A promising approach to building and evolving large-scale and long-lived tactical information management systems are standards-based QoS-enabled publish/subscribe (pub/sub) platforms that enable applications to communicate by publishing information they have and subscribing to information they need in a timely manner. Since there is little existing evaluation of how well these platforms meet the performance needs of tactical information management, this paper provides two contributions: (1) it describes three common architectures for the OMG Data Distribution Service (DDS), which is a QoS-enabled pub/sub platform standard, and (2) it evaluates three implementations of these architectures to investigate their design tradeoffs and to compare their performance. Our results show that DDS implementations perform well in general and are well-suited for certain classes of data-critical tactical information management systems.

Keywords: Tactical Information Management; QoS-enabled Pub/Sub Platforms; Data Distribution Service

1. INTRODUCTION

Mission-critical tactical information management systems run increasingly often in net-centric environments that are characterized by thousands of platforms, sensors, decision nodes, and computers connected together to exchange information, support collaborative decision making, and effect changes in the physical environment. For example, the Global Information Grid (GIG) [11] is designed to ensure that the right information gets to the right place at the right time by satisfying end-to-end quality of service (QoS) requirements, such as latency, jitter, throughput, dependability, and scalability. Tactical information management systems often have many applications per computing node, where the failure of one application should not degrade other applications. These applications often require point-to-multipoint communication mechanisms to distribute data from suppliers to multiple consumers. There are often multiple high-priority and low-priority datastreams running in parallel, where the high-priority data must pre-empt low-priority data.

A promising infrastructure for such tactical information management systems is QoS-enabled publish/subscribe (pub/sub) middleware that provides:

- Location-independent, universal access to information from a wide variety of sources running over a multitude of hardware/software platform and network deployments.
- An orchestrated information environment that aggregates, filters, and prioritizes the delivery of this information to work effectively under the restrictions of transient and enduring computing and communication resource constraints.
- Continuous adaptation to changes in the operating environment, such as dynamic network topologies, publisher and subscriber membership changes, reprioritization in information importance, and intermittent connectivity.
- Various QoS parameters and mechanisms that enable applications and administrators to customize the way information is delivered, received, and processed in the appropriate form and level of detail to users at multiple levels in a tactical information management system.

Conventional Service-Oriented Architecture (SOA) middleware platforms have had limited success in providing these capabilities, due to their lack of support for data-centric QoS mechanisms. For example, the Java Messaging Service for Java 2 Enterprise Edition (J2EE) is a SOA middleware platform that is not well-suited for tactical information management environments due to its limited QoS support, lack of real-time operating system integration, and high

---

[†] This work was sponsored in part by the AFRL/IF Pollux project and Vanderbilt University's Summer Undergraduate Research program.

time/space overhead. Even conventional QoS-enabled SOA middleware, such as Real-time CORBA [9], is poorly suited for dynamic data dissemination between many publishers and subscribers due to excessive layering, extra time/space overhead, and inflexible QoS policies.

To address these limitations—and to better support tactical information management—the OMG Data Distribution Service (DDS) [6] specification has emerged as a standard for QoS-enabled pub/sub communication aimed at mission-critical tactical information management systems. It is designed to provide (1) *location independence* via anonymous pub/sub protocols that enable communication between collocated or remote publishers and subscribers, (2) *scalability* by supporting large numbers of topics, data readers, and data writers, and (3) *platform portability and interoperability* via standard interfaces and transport protocols. Multiple implementations of DDS are now available, ranging from high-end COTS products [4] to open-source community-supported projects. DDS is used in a wide range of distributed real-time and embedded (DRE) systems, including traffic monitoring [14], control of unmanned vehicle communication with ground stations [16], and semiconductor fabrication devices [15].

Although DDS is designed to be scalable, efficient, and predictable, few researchers have evaluated and compared the performance of DDS implementations empirically for common tactical information management scenarios. This paper addresses this gap in the literature by describing the results of the Pollux project, which is an ongoing R&D activity aimed at evaluating a range of pub/sub platforms to compare how their architecture and design features affect their performance and suitability for tactical information management. This paper also describes the design and application of an open-source DDS benchmarking environment we developed in Pollux to automate the comparison of pub/sub latency, jitter, throughput, and scalability.

The remainder of this paper is organized as follows: Section 2 summarizes the DDS specification and the architectural differences of three popular DDS implementations; Section 3 describes the hardware configurations of our testbed and introduces an open-source DDS Benchmark Environment (DBE); Section 4 analyzes the results of benchmarks conducted using DBE; Section 5 compares our work with related research on performance evaluation of pub/sub platforms; and Section 6 presents concluding remarks.

2. OVERVIEW OF DDS

2.1 Core Features and Benefits of DDS

The OMG Data Distribution Service (DDS) specification provides a data-centric communication standard for a range of DRE computing environments, from small networked embedded systems to large-scale information backbones. At the core of DDS is the *Data-Centric Publish-Subscribe* (DCPS) model, whose specification defines standard interfaces that enable applications running on heterogeneous platforms to write/read data to/from a virtual global data space in a DRE system. Applications willing to share information can use this data space to declare their intent to publish data that is categorized into one or more topics of interest to others. Similarly, applications that are interested in certain topics can use the data space to declare their intent to become subscribers and access the data.

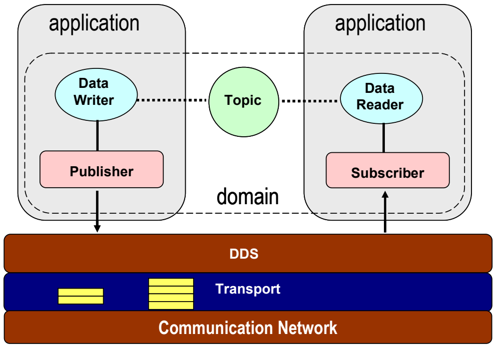


Fig. 1. Architecture of DDS

The underlying DCPS middleware propagates data samples written by publishing applications into the global data space, where they are disseminated to subscribing applications [6]. The DCPS model decouples the declaration of information access intent from the information access itself [4], thereby enabling the DDS middleware to support and optimize QoS-enabled communication. As shown in Fig. 1, a canonical DCPS model is comprised of the following elements that provide functionalities for a DDS application to publish/subscribe to data samples of interest.

- **Domain**. DDS applications send and receive data within a Domain. A Domain is a virtual space that connects certain publishing and subscribing applications. Only applications within the same domain can communicate, and this restriction helps isolate and optimize communication within a community that shares common interests. Note that although only one domain is shown in Fig. 1, a system can be divided into as many domains as needed to meet system requirements.
- **Entity**. Within a domain, DDS defines an abstract element called Entity, which contains a few generic operations that it passes to the elements listed below that specialize it. All entities have associated QoS policies, initialized to default values unless explicitly modified.
- **DomainParticipant.** Created by a singleton factory, a *DomainParticipant* is the application's access point to a Domain. Applications use DomainParticipants to create Topics, Publishers, and Subscribers, which are described next.
  - **Publisher.** A *Publisher* creates and manages one or more *DataWriter* entities
  - **Subscriber.** A *Subscriber* creates and manages one or more *DataReader* entities..
  - **DataWriter.** A *DataWriter* is the actual object used to send data samples, and is always associated with a particular *Topic*.
  - **DataReader**. A *DataReader* is the actual object used to receive data samples, and is always associated with a particular *Topic*.
  - **Topic**. A *Topic* consists of a data type and a name, and it connects a *DataWriter* with a *DataReader*. Data samples start flowing only when the Topic associated with a DataWriter matches the Topic associated with a DataReader.

The DCPS middleware layer is responsible for marshaling/de-marshaling and sending/receiving the data to/from the virtual global data space using standard DDS transports, such as UDP or other protocols like shared memory. Applications simply use the DDS entities outlined above to read/write data from/to the global data space without having to wrestle with low-level implementation details, and without having to know which, or how many, entities are at the other end of the data transfer.

Compared with conventional client/server-based SOA middleware that is designed to support the requirements of business systems, DDS is data-oriented and designed to support applications with DRE QoS requirements. Since DDS focuses on data rather than object interfaces, the DDS standard can be implemented in a more flexible way to make the data transmission more efficient, *e.g.*, it can maximize throughput and minimize latency and jitter in a tactical network environment.

For example, unlike CORBA pub/sub services such as its Event and Notification Services, DDS does not implement its capabilities as a layer built on top of an object request broker, which helps reduce transmission latency and jitter. Another DDS capability that distinguishes it from conventional SOA middleware is its support of nearly two dozen QoS policies to control many aspects of data delivery and quality, including:

- The lifetime of each data sample, *i.e.*, whether the data is destroyed after being sent, kept available during the publisher's lifetime, or allowed to stay persistent for a specified duration after the publisher shuts down.
- The degree and scope of coherency for information updates, *i.e.*, whether a group of updates can be received as a unit and in the order in which they were sent.
- The frequency of information updates, *i.e.*, the rate at which updated values are sent or received.
- The maximum latency of data delivery, *i.e.*, a bound on the acceptable interval between the time data is sent and the time it is received
- The priority of data delivery, *i.e.*, the priority used by the underlying transport to deliver the data.
- The reliability of data delivery, *i.e.*, whether missed deliveries will be retried.
- How to arbitrate simultaneous modifications to shared data by multiple writers, *i.e.*, to determine which modification to apply.
- Mechanisms to assert and determine liveliness, *i.e.*, whether or not a publish-related entity is active.
- Parameters for filtering by data receivers, *i.e.*, determine which data values are accepted and which are rejected.
- The duration of data validity, *i.e.*, the specification of an expiration time for data to avoid delivering "stale" data.

- The depth of the 'history' included in updates, *i.e.*, how many prior updates will be available at any time, *e.g.*, 'only the most recent update,' 'the last *n* updates,' or 'all prior updates'.

These DDS QoS policies can be configured at various levels of granularity (i.e., topics, publishers, data writers, subscribers, and data readers), thereby allowing application developers to construct customized contracts based on the specific QoS requirements of individual use cases. Since the identity of publishers and subscribers are unknown to each other, the DDS middleware is responsible for determining whether QoS policies offered by a publisher are compatible with those required by a subscriber, allowing data distribution only when compatibility is satisfied.

2.2 Overview of DDS Implementation Architectures

As outlined in Section 2.1, the DDS specification defines a wide range of QoS policies and interfaces used to exchange data samples between entities. The specification intentionally does not address how to implement the services or manage DDS resources internally, so DDS providers are free to innovate. Naturally, the communication models, distribution architectures, and implementation techniques used by DDS providers significantly impact application behavior and QoS, *i.e.*, different choices affect the suitability of DDS implementations and configurations for various types of tactical information management applications.

Table 1. Supported DDS Communication Models

Impl	Unicast	Multicast	Broadcast
DDS1	Yes (default)	Yes	No
DDS2	No	Yes	Yes (default)
DDS3	Yes (default)	Yes	No

By design, DDS specification allows implementations and applications to take advantage of various communication models, such as unicast, multicast, and broadcast transports. The communication models supported for the three DDS implementations we evaluated are shown in Table 1 (the specific DDS product names are "shrouded" pending final approval from the companies that produce them). DDS1 and DDS3 support unicast and multicast, whereas DDS2 supports multicast and broadcast. These DDS implementations all use layer 3 network interfaces (IP multicast and broadcast) to handle the network traffic for different communication models, rather than more scalable multicast protocols, such as Richocet [5], which combine native IP group communication with proactive forward error correction to achieve high levels of consistency with stable and tunable overhead. Each DDS implementation has a different architectural design, as described in the remainder of this section.

2.2.1 Federated Architecture

The federated DDS architecture shown in Fig. 2 uses a separate daemon process for each network interface. These daemons are typically started when the nodes are booted and must be initialized before entities in the domain can communicate.

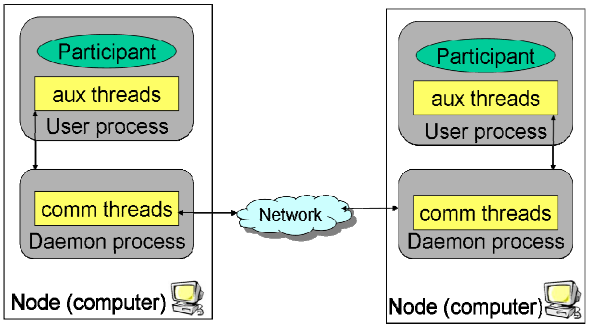


Fig. 2. Federated DDS Architecture

Once started, each daemon communicates with others and establishes data channels based on reliability requirements (*e.g.*, reliable or best-effort), importance (i.e. priority), urgency (i.e. latency-budget) and transport mode (*e.g.*, broadcast or multicast inclusive traffic shaping and reactivity).. Each channel handles communication and QoS for all the entities requiring its particular properties. Using a daemon process decouples the entities (which run in a separate user process) from configuration- and communication-related details. For example, the daemon process can use a configuration file to store common system parameters shared by communication endpoints associated with a network interface, so that changing the configuration does not affect application code or processing.

The advantage of a federated architecture is that applications can support a larger number of DDS entities on the same node, *e.g.*, by bundling messages that originate from collocated entities. Using a separate daemon process to mediate access to the network can also help to (1) simplify application configuration of policies for a group of entities associated with the same network interface, (2) provide a network scheduler that prioritize messages from different communication channels, and (3) prevent faulty applications from generating uncontrolled or excessive network-traffic. One disadvantage of a daemon-based federated architecture is that it introduces configuration steps that must be managed separately from applications to ensure proper functioning and avoid single points of failure. Moreover, applications must cross extra process boundaries to communicate, which can introduce overhead that increases latency and jitter.

2.2.2 Decentralized Architecture

The decentralized DDS architecture shown in Fig. 3 places the communication- and configuration-related capabilities into the same process as the application itself. These capabilities execute in separate threads (rather than in a separate process) and are used by the middleware to handle communication and QoS.

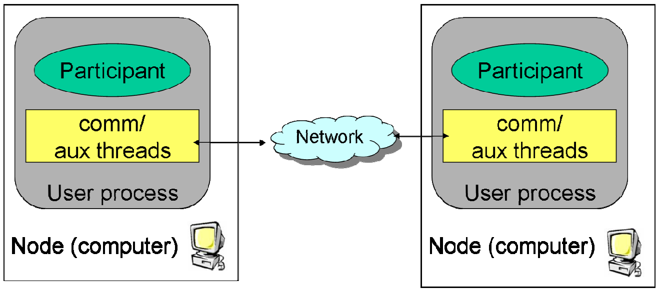


Fig. 3. Decentralized DDS Architecture

The advantage of a decentralized architecture is that each application is self-contained, without the need of a separate daemon. As a result, latency and jitter are reduced because fewer context switches are involved compared to the federated architecture, and there is one less configuration and failure point. A disadvantage, however, is that specific configuration details, such as multicast address, port number, reliability model, and parameters associated with different transports, must be defined at the application level. Requiring each application developer to handle these details is tedious, error-prone, and potentially non-portable. This architecture also makes it hard to buffer data sent between multiple DDS applications on a node, and thus does not provide the same entity-per-node scalability benefits offered by the federated architecture.

2.2.3 Centralized Architecture

The centralized architecture shown in Fig. 4 uses a single daemon server running on a designated node to store the information needed to manage topics and connections. The data itself passes directly from publishers to subscribers, but the control and initialization activities (such as data type registration, topic creation, and QoS value assignment, modification and matching) require communication with this server.

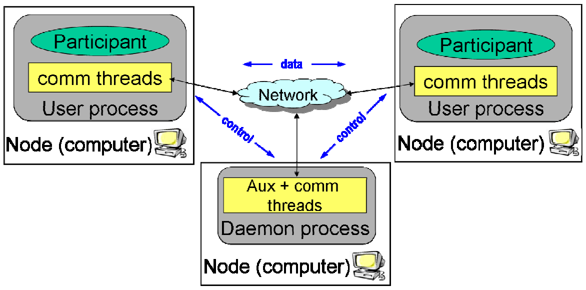


Fig. 4. Centralized DDS Architecture

The advantage of the centralized approach is its simplicity of implementation and configuration since all control information resides in a single location. The disadvantage is that the daemon is a single point of failure, as well as a potential performance bottleneck in a heavily loaded system.

The remainder of this paper investigates how the architecture differences described above can affect the performance experienced by certain types of tactical information management applications, *i.e.*, those that transmit small amounts of data in a point-to-point manner.

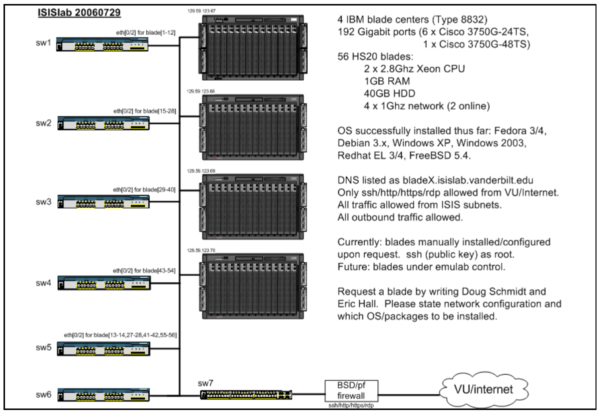
3.METHODOLOGY FOR DDS IMPLEMENTATION EVALUATION

This section describes our methodology for evaluating DDS implementations to determine how well they support certain classes of tactical information management applications, particularly those that generate small amounts of data periodically, which require low latency and jitter.

3.1 Benchmarking Environment

Hardware and Software Infrastructure

The computing nodes we used to run our experiments are hosted on ISISlab [19], which is a cluster of computers and network switches that can be arranged in many configurations, as shown in Fig. 5. Each computer used in our tests contained the following hardware configuration: dual 2.8 GHz Xeon CPUs, 1GB of ram, 40GB HDD, and gigabit Ethernet cards. To ensure system stability and minimum operating system jitter, real-time Fedora Core 4 Linux kernels were installed on each computer and the machines were isolated from the rest of the network throughout the duration of each test. In addition, all processes were run as root and executables used the Linux real-time scheduling class to further leverage features provided by the real-time kernel.



4 IBM blade centers (Type 8832)
192 Gigabit ports (6 x Cisco 3750G-24TS,
1 x Cisco 3750G-48TS)
56 HS20 blades:
2 x 2.8Ghz Xeon CPU
1GB RAM
40GB HDD
4 x 1Ghz network (2 online)

OS successfully installed thus far: Fedora 3/4, Debian 3.x, Windows XP, Windows 2003, Redhat EL 3/4, FreeBSD 5.4.

DNS listed as bladeX.isislab.vanderbilt.edu
Only ssh/http/https/rdp allowed from VU/Internet.
All traffic allowed from ISIS subnets.
All outbound traffic allowed.

Currently: blades manually installed/configured upon request. ssh (public key) as root.
Future: blades under emulab control.

Request a blade by writing Doug Schmidt and Eric Hall. Please state network configuration and which OS/packages to be installed.

Fig. 5: ISISlab structure

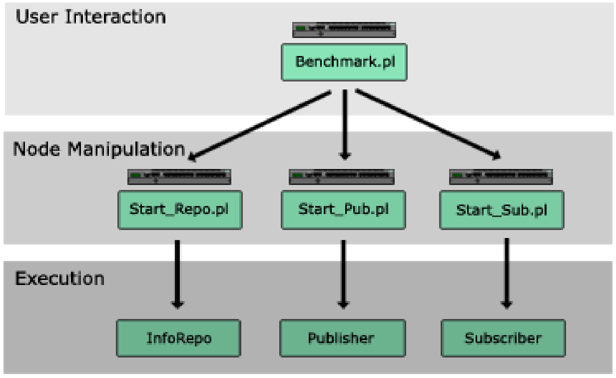


Fig. 6. DDS Benchmarking Environment (DBE)

DDS Benchmark Environment (DBE)

Achieving good coverage of a test space where parameters can vary in several orthogonal dimensions leads to a combinatorial explosion of test types and configurations. Manually running tests for each configuration and each middleware implementation on each node is tedious, error-prone, and time-consuming. The task of managing and organizing test results also grows exponentially along with the number of distinct test configuration combinations.

To facilitate the growth of our tests both in variety and complexity, we created the *DDS Benchmarking Environment* (DBE), which is an open-source framework for automating our DDS testing. The DBE consists of (1) a repository that contains scripts, configuration files, test ids, and test results, (2) a hierarchy of Perl scripts to automate test setup and execution, and (3) a shared library for gathering results and calculating statistics.

Our efforts to streamline test creation, execution and analysis are ongoing. As shown in Fig. 6, the DBE currently has three levels of execution designed to enhance flexibility, performance, and portability, while incurring low overhead. Each level of execution has a specific purpose: the top level is the user interface, the second level manipulates the node itself, and the bottom level is comprised of the actual executables (*e.g.*, publishers and subscribers for each DDS implementation). DBE runs all test executables locally, eliminating the effects on network traffic due to DBE test artifacts.

3.2 Evaluation Metrics

Our evaluations in this paper compare the performance of the C++ implementations of DDS shown in Table 2 against each other using micro-benchmarks in the ISISlab environment described in Section 3.1.

Table 2: DDS Implementations Tested

Impl	Version	Distribution Architecture
DDS1	4.1d	Decentralized Architecture
DDS2	2.2.5	Federated Architecture (with "Direct-Write" optimization)[‡]
DDS3	0.11	Centralized Architecture

We compare the performance of these pub/sub mechanisms by using the following metrics:

- **Latency**, which is defined as the roundtrip time between the sending of a message and reception of an acknowledgment from the subscriber. In our test, the roundtrip latency is calculated as the average value of 10,000 round trip measurements.
- **Jitter**, which is the standard deviation of the latency.

As part of the ongoing AFRL/IF Pollux project, we are enhancing the DBE to evaluate other interesting features of DDS needed by large-scale tactical information management systems, including:

- Benchmarking other performance metrics for the various DDS implementations, including 1-to-n latency, jitter, and throughput for reliable and best-effort communication, as well as CPU and memory utilization.
- Tailoring our DBE benchmarks to explore key classes of applications in tactical information management systems, including periodic sensor processing, track processing systems, and asynchronous alert systems.
- Empirically evaluating a wider range of QoS configurations, *e.g.* durability, reliable vs. best-effort, and integration of durability, reliability and history depth,
- Measuring discovery time for various entities,

These results will appear in our Pollux project website at www.dre.vanderbilt.edu/DDS as they are completed, along with the source code for DBE.

4. EMPIRICAL RESULTS

This section analyzes the results of our initial benchmarks conducted using the DBE on ISISlab. We evaluate point-to-point (*i.e.*, 1-to-1) roundtrip latency performance of DDS implementations within a single node, as well as between two distributed nodes. To ensure optimal product performance, configurations for each DDS implementation were tuned carefully based on extensive discussion with the DDS vendors.

Benchmark design. Latency is an important measurement to evaluate tactical information management performance. Our test code measures roundtrip latency for each DDS implementation described in Section 3.1. The IDL structure for our benchmark test is shown below.

```
const short MAX_MSG_LENGTH = 16384;
struct PubMessage {
  long seqnum;
  sequence<octet, MAX_MSG_LENGTH> data;
};
struct AckMessage  { long seqnum; }
```

[‡] DDS2 applies an internode optimization called "Direct-Write" that enables publishers to write the data directly to the subscriber daemon rather than going through a daemon on the publisher node, thereby helping to reduce latency relative to a pure Federated architecture described in Section 2.2.1.

The DataWriter object in the publisher writes an octet sequence of a designated payload size, which ranges from 4 bytes to 16,384 bytes by powers of 2. When the DataReader in the subscriber receives the data it replies to the publisher with a 4-byte application-level acknowledgement. We use this "request/response" protocol to ensure that the round-trip timestamp is recorded on the publisher node, thereby eliminating clock skew problems. Since DDS traffic typically uses 'one way' communication from publisher to subscriber(s) without any application-level acknowledgements, however, this request/response protocol make the performance look somewhat worse that would actually occur in practice.

The publisher test code measures latency by timestamping the data transmission and subtracting that from the timestamp value it receives in the ack message from subscriber. This test evaluates how fast data is transferred from one node to another at different payload sizes. To ensure that our benchmark applications will be in a steady state when collecting statistical data, we send primer samples to "warm up" the applications before actually measuring the data. This warm-up period allows time for possible discovery activity related to other subscribers to finish, and for any other first-time actions, on-demand actions, or lazy evaluations to be completed, so that their extra overhead does not affect the statistics calculations. We also use the Linux real-time scheduling class in our tests to minimize jitter.

Analysis of Results. Fig. 7 and Fig. 8 compare latency/jitter results for simple sequence types running on a single node. Fig. 9 and Fig. 10 compare latency/jitter results for simple sequence types running on multiple nodes. As discussed in Section 2.1, DDS is data-oriented rather than object-oriented, which helps explain why DDS implementations have good overall performance, *e.g.*, their latency and jitter are within the bounds expected by most DRE systems.

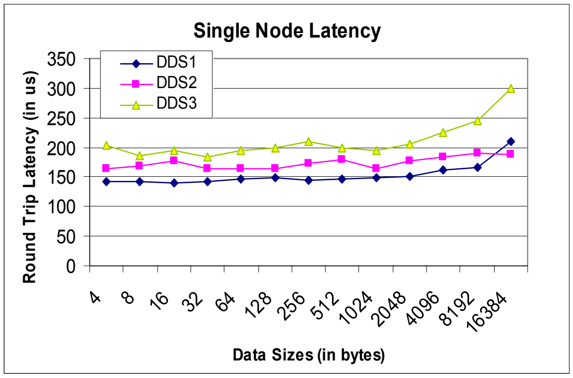


Fig. 7. Single Node Latency

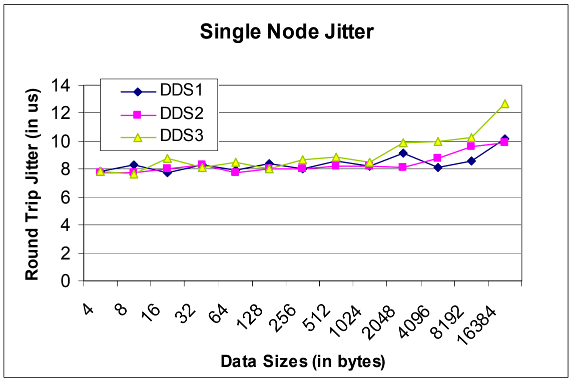


Fig. 8. Single Node Jitter

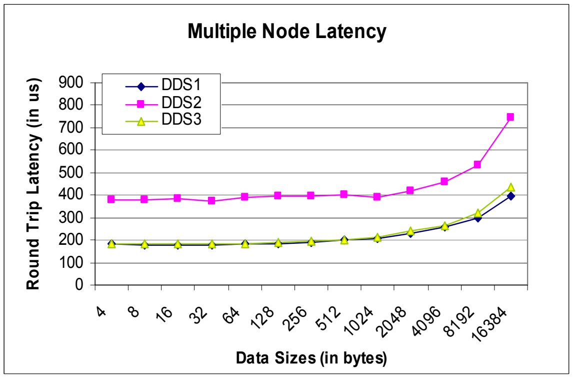


Fig. 9. Multiple Node Latency

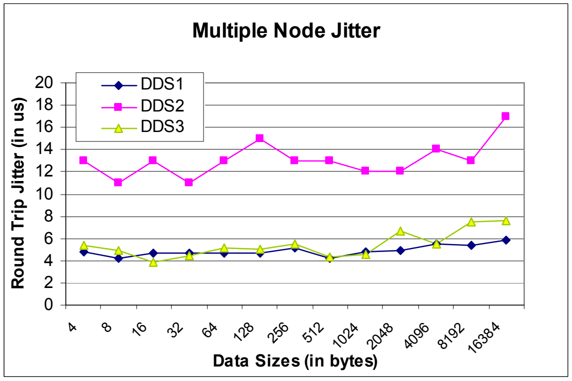


Fig. 10. Multiple Node Jitter

DDS1 and DDS2 both perform better than DDS3 on same-node latencies because they use a shared memory transport, whereas DDS3 uses UDP loopback. Despite the lack of shared memory usage, single node jitter for DDS3 is well-paced with its competitors, though eventually tapering off with larger payloads (see Fig. 8 from 2k onward.) Further investigation is needed for same-node round trip latency when data payloads exceed 16k to see if the DDS1 and DDS3 latencies continue to grow while those of DDS2 remain constant.

From our preliminary analysis outlined in Section 2, we expect DDS2 to perform better with more DDS entities per node. Moreover, increases in data size are less likely to affect DDS2 performance on same-node communication since DDS2 eliminates the marshaling/de-marshaling that occurs during the same operations on DDS1. The single DDS2 daemon per-network interface allows these types of optimizations, since byte ordering on the same machine is extremely unlikely to change.

The results for the multiple node tests vary more than the single-node tests since the latency and jitter graphs show a wider difference between the DDS1 and DDS3 latency on one hand and DDS2 latency on the other. DDS1 and DDS3 have lower latency because their architectures are optimized for direct communication between Publishers and Subscribers. DDS2, in contrast, is designed for DRE system configurations where many DDS application processes run on each node, thereby leveraging the scalability of its federated architecture. A consequence of its federated architecture, however, is the higher latency when moving DDS data from the DataWriter on the Publisher node to the daemon and DataReader on the Subscriber node. The "Direct-Write" optimization described in Section 3.2 helps minimize this overhead, but extra context switching, synchronization, and data copying are still incurred.

Although our results above demonstrated the impact of DDS architectural choices on certain performance characteristics, it is important to note that tactical information management applications typically require much more than low point-to-point latency. In particular, they also require advanced QoS capabilities, such as traffic-shaping, priority-banding, network partitioning, and data bundling. Systematically evaluating the impact of different DDS architectures on these capabilities motivates the need for enhancements to the DBE described in Section 3.2. Our future work on DDS will also include

- Devising generators that can emulate various workloads and use cases for various types of tactical information management systems, including periodic sensor processing, track processing systems, and asynchronous alert systems
- Identifying scenarios that distinguish performance of QoS policies and features (*e.g.*, collocation of applications), and
- Evaluating the suitability of DDS in heterogeneous dynamic environments, *e.g.*, mobile ad hoc networks, where system resources are limited and dynamic topology, domain and entity changes are common.

5. RELATED WORK

To support emerging tactical information management systems, pub/sub middleware in general, and DDS in particular, have attracted an increasing number of research efforts (such as COBEA [20] and Siena [12]) and commercial products and standards (such as JMS [10], WS_NOTIFICATION [13], and the CORBA Event and Notification services [17]). This section describes several projects that are related to the work presented in this paper.

Open Architecture Benchmark. Open Architecture Benchmark (OAB) [8] is a DDS benchmark effort associated with the Open Architecture Computing Environment, an open architecture developed by the US Navy. Joint efforts have been conducted in OAB to evaluate DDS products, in particular DDS1 and DDS2, to understand the ability of these DDS products to support the bounded latencies required by naval systems. Their results indicate that both products perform quite well and meet the requirements of typical naval systems. Our DDS work extends that effort by (1) including DDS3 in the comparisons and (2) classifying the different architectures used by these implementations, and offering explanations of performance results by referring to these differences.

S-ToPSS. There has been an increasing demand for content-based pub/sub applications, where subscribers can use a query language to filter the available information, and receive only a subset of the data that is of interest. Most solutions support only syntactic filtering, *i.e.*, matching based on syntax, which greatly limits the selectivity of the information. In [7] the authors investigated how current pub/sub systems can be extended with semantic capabilities, and proposed a prototype of such middleware called the *Semantic - Toronto Publish/Subscribe System* (S-ToPSS). For a highly intelligent semantic-aware system, simple synonym transformation is not sufficient. S-ToPSS extends this model by adding another two layers to the semantic matching process, *concept hierarchy* and *matching functions*. Concept hierarchy makes sure that events (data messages, in the context of this paper) that contain generalized filtering information do not match the subscriptions with specialized filtering information, and that events containing more specialized filtering than the subscriptions will match. Matching functions provide a many-to-many structure to specify more detailed matching relations, and can be extended to heterogeneous systems. DDS also provides QoS policies that support content-based filters for selective information subscription, but they are currently limited to syntactic match. Our future work will explore the possibility of introducing semantic architectures into DDS and evaluate their performance.

PADRES. The Publish/subscribe Applied to Distributed Resource Scheduling (PADRES) [1] is a distributed, content-based publish/subscribe messaging system. A PADRES system consists of a set of brokers connected by an overlay network. Each broker in the system employs a rule-based engine to route and match publish/subscribe messages, and is used for composite event detection. PADRES is intended for business process execution and business activity monitoring, rather than for DRE systems. While not conforming to the DDS API, its publish/subscribe model is close to that of DDS, so we plan to explore how a DDS implementation might be based on PADRES.

6.CONCLUDING REMARKS

This paper described the architectures of three implementations of the OMG Data Distribution Service (DDS). DDS is particularly relevant for tactical information management since (1) its communication model provides a range of QoS parameters that allow applications to control many aspects of data delivery in a network, (2) its implementations can be optimized heavily for various real world scenarios, and (3) DDS implementations can be configured to leverage fast transports, *e.g.*, using shared memory to minimize data copies within a single node, and to improve scalability, *e.g.*, by using multicast to communicate between nodes.

We then presented the DDS Benchmarking Environment (DBE) and showed how we used the DBE to compare the performance of these DDS implementations for point-to-point latency and jitter of a simple datatype. Based on our test results, experience developing the DBE, and numerous DDS experiments, we learned the following lessons: (1) DDS holds great promise for DRE systems, (2) architectural differences in DDS implementations produce a line of products with varying specialties that appeal to different types of applications and real time scenarios, and (3) DDS vendors are working diligently to improve their products and innovate within the constraints of a standard specification.

All the source code for the DBE and DDS tests described in this paper are available in open-source format at www.dre.vanderbilt.edu/DDS.

Acknowledgements

We would like to thank Real-Time Innovations (RTI), Prism Technologies (PT), and Object Computing Inc. (OCI), particularly Gerardo Pardo-Castellote and Gong Ke from RTI, Hans van't Hag and Reinier Torenbeek from PrismTech, and Yan Dai, Adam Mitz, and Scott Harris from OCI for their help in improving the experiments reported in this paper.

REFERENCES

1. A. Cheung, H. Jacobsen, "Dynamic Load Balancing in Distributed Content-based Publish/Subscribe," ACM/IFIP/USENIX Middleware 2006, December, Melborne, Australia.
2. D. C. Schmidt and C. O'Ryan, "Patterns and Performance of Distributed Real-time and Embedded Publisher/Subscriber Architectures," Journal of Systems and Software, Special Issue on Software Architecture -- Engineering Quality Attributes, edited by Jan Bosch and Lars Lundberg, Oct 2002.
3. C. Gill, J. M. Gossett, D. Corman, J. P. Loyall, R. E. Schantz, M. Atighetchi, and d. C. Schmidt, "Integrated Adaptive QoS Management in Middleware: An Empirical Case Study," Proceedings of the 10th Real-time Technology and Application Symposium, May 25-28, 2004, Toronto, CA.
4. G. Pardo-Castellote, "DDS Spec Outfits Publish-Subscribe Technology for GIG," COTS Journal, April 2005.
5. M. Balakrishnan, K. Birman, A. Phanishayee, and Stefan Pleisch, "Ricochet: Low-Latency Multicast for Scalable Time-Critical Services," Cornell University Technical Report, www.cs.cornell.edu/ projects/quicksilver/ public_pdfs/ricochet.pdf.
6. OMG, "Data Distribution Service for Real-Time Systems Specification," www.omg.org/docs/formal/04-12-02.pdf.
7. I. Burcea, M. Petrovic, H. Jacobsen, "S-ToPSS: Semantic Toronto Publish/Subscribe System," International Conference on Very Large Databases (VLDB). p. 1101-1104. Berlin, Germany, 2003.
8. B. McCormick, L. Madden, "Open Architecture Publish-Subscribe Benchmarking," OMG Real-Time Embedded System Work Shop 2005, www.omg.org/news/meetings/workshops/RT_2005/03-3_McCormick-Madden.pdf.
9. A. S. Krishna, D. C. Schmidt, R. Klefstad, and A. Corsaro, "Real-time CORBA Middleware," in *Middleware for Communications*, edited by Qusay Mahmoud, Wiley and Sons, New York, 2003.
10. Hapner, M., Burridge, R., Sharma, R., Fialli, J., and Stout, K. 2002. Java Message Service. Sun Microsystems Inc., Santa Clara, CA.

11. Fox, G., Ho,A., Pallickara, S., Pierce, M., and Wu,W, "Grids for the GiG and Real Time Simulations," Proceedings of Ninth IEEE International Symposium DS-RT 2005 on Distributed Simulation and Real Time Applications, 2005.
12. D.S.Rosenblum, A.L.Wolf, "A Design Framework for Internet-Scale Event Observation and Notification," *6th European Software Engineering Conference*. Lecture Notes in Computer Science 1301, Springer, Berlin, 1997, pages 344-360.
13. S. Pallickara, G. Fox, "An Analysis of Notification Related Specifications for Web/Grid Applications," International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume II pp. 762-763.
14. Real-Time Innovation, "High-reliability Communication Infrastructure for Transportation," www.rti.com/markets/transportation.html.
15. Real-Time Innovation, "High-Performance Distributed Control Applications over Standard IP Networks," www.rti.com/markets/industrial_automation.html.
16. Real-Time Innovation, "Unmanned Georgia Tech Helicopter files with DDS1," controls.ae.gatech.edu/gtar/2000review/rtiDDS1.pdf.
17. P. Gore, D. C. Schmidt, C. Gill, and I. Pyarali, "The Design and Performance of a Real-time Notification Service," Proceedings of the 10th IEEE Real-time Technology and Application Symposium (RTAS '04), Toronto, CA, May 2004.
18. N. Abu-Ghazaleh, M. J. Lewis, and M. Govindaraju, "Differential Serialization for Optimized SOAP Performance**,**" Proceedings of *HPDC-13: IEEE International Symposium on High Performance Distributed Computing*, Honolulu, Hawaii, pp. 55-64, June 2004.
19. Distributed Object Computing Group, "DDS Benchmark Project," www.dre.vanderbilt.edu/DDS.
20. C.Ma and J.Bacon "COBEA: A CORBA-Based Event Architecture," In Proceedings of the 4rd Conference on Object-Oriented Technologies and Systems, USENIX, Apr.1998
21. P. Gore, R. K. Cytron, D. C. Schmidt, and C. O'Ryan, "Designing and Optimizing a Scalable CORBA Notification Service," in Proceedings of the Workshop on Optimization of Middleware and Distributed Systems, (Snowbird, Utah), pp. 196–204, ACM SIGPLAN, June
22. Steve Vinoski, CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments, IEEE Communications Magazine, February, 1997.