TWINOAKS COMPUTING INC.
PRACTICAL MIDDLEWARE EXPERTISE

What can DDS do for You?

Learn how dynamic publish-subscribe messaging can improve the flexibility and scalability of your applications.

Contents:

Abstract

The Data Distribution Service (DDS) technology may not be as well known or understood as other middleware communications technologies like Corba, JMS, Web Services or even traditional sockets that are key components of nearly all distributed applications. Developers everywhere are familiar with these conventional data communications solutions, and may be comfortable customizing them to their particular domain and application. However, the flexibility and configurability of DDS makes it a better solution for applications in a wide variety of industries, from enterprise servers to deeply embedded applications.
This paper focuses on the DDS technology in embedded environments and compares it with Sockets, Corba, JMS, and SOA Web Services.

What does DDS do?

DDS, the Data Distribution Service, is a data communications standard managed by the OMG (http://www.omg.org) that describes low-latency data communications for distributed applications. The DDS standard includes support for Type-Safe application defined data types; Dynamic Discovery of publishers, subscribers, and topics; rich Quality of Service policy configuration; and on the wire Interoperability. DDS implementations provide high-performance data communications, suitable for real-time and near real-time systems. There are currently several commercial and open source implementations of the DDS standard available for use, including products built for and targeting embedded communications.

The DDS Standard contains an easy to use, well defined Application Programming Interface (API). This allows the developer to write *portable* code, code that will work with any compliant DDS implementation. The DDS standard references the Real Time Publish Subscribe (RTPS) Wire Protocol standard which defines the wire protocol for DDS communications. This allows applications built with different DDS implementations to communicate, or *interoperate*, with each other. Users of DDS do not tie themselves to a particular vendor, but to a standard, and can change or intermix DDS vendors throughout the development and deployment cycles.
In general, DDS is a peer-to-peer communication model requiring no gateways, servers, or daemons that must be run or configured.

Each application that will communicate over DDS contains the DDS API and provides the discovery, and other required communication details.

Historically, DDS has been used in large DoD systems to satisfy Open Architecture requirements for Extensibility, Maintainability, Composability, and Interoperability, but only in the larger computer components of these systems. Now, with the availability of small-footprint DDS implementations, many other applications can benefit from standardized publish subscribe communications. New areas starting to adopt DDS include safety critical applications like UAVs; renewable energy and smart grid suppliers; and resource constrained environments within the DoD where enterprise DDS solutions were unsuitable.

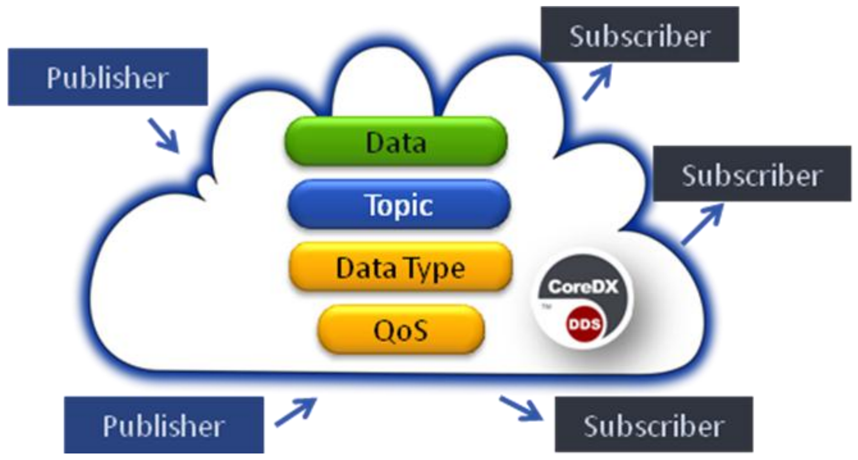


Figure 1

The Strengths of DDS

Publish Subscribe Architecture

DDS provides a flexible publish subscribe architecture.   A publish subscribe architecture promotes a loose coupling between data architecture is flexible and dynamic; it is easy to adapt and extend DDS based systems to changing environments and requirements. The following figure illustrates the DDS Publish Subscribe architecture where multiple Publishers and Subscribers exchange strongly typed data through a common Topic. The communications are controlled by a robust Quality of Service model.

An application may be a publisher of data, a subscriber of data, or both a publisher and subscriber. An application may participate in multiple Domains and include multiple publishers and subscribers. Figure 2 is an example of how DDS might be applied in a system. This example has several sources of "raw data", a data processor that performs some processing on the raw data to produce "processed data", several end users working with the processed data, and an administrative user performing analysis, maintenance, or auditing functions.

In this example, the darker blue boxes represent applications communicating over a DDS network. These applications might be running together on 1 host, or they might be distributed over multiple hosts. A DDS application simply publishes or subscribes to data, without concern for what, if anything, might be on the other end of its communications.

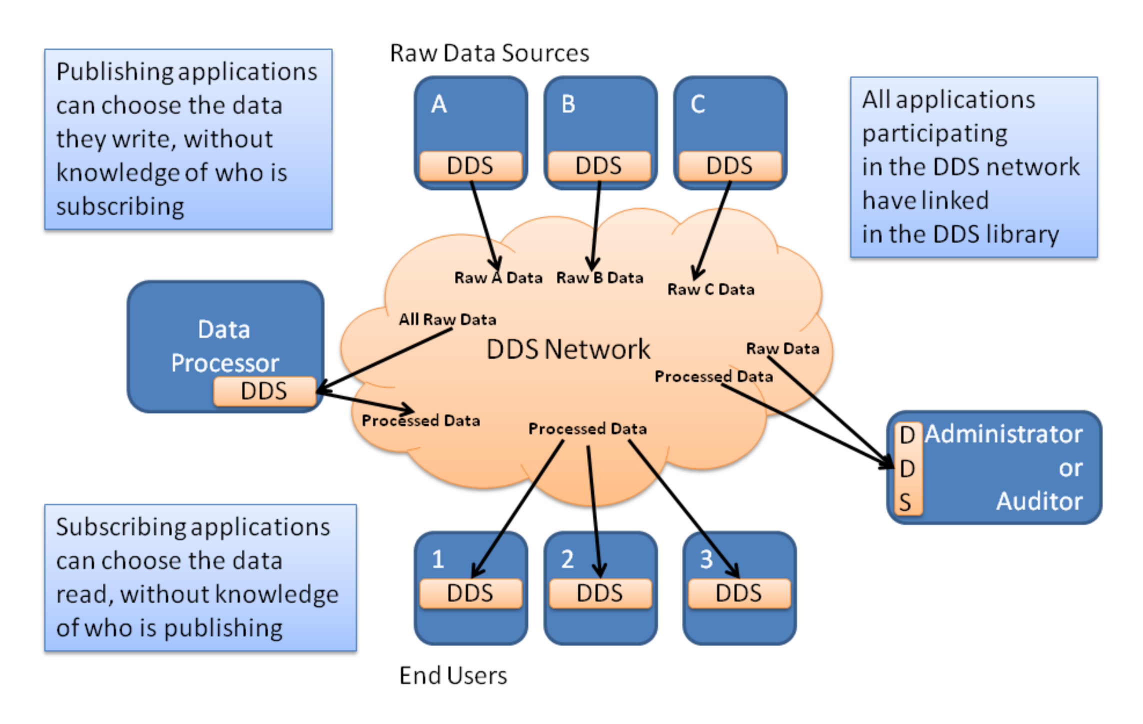

Publishing applications can choose the data they write, without knowledge of who is subscribing
Raw Data Sources
A
B
C
DDS
All applications participating in the DDS network have linked in the DDS library
Raw A Data
Raw B Data
Raw C Data
All Raw Data
Data Processor
DDS Network
Raw Data
Processed Data
Processed Data
Processed Data
DDS Administrator or Auditor
Subscribing applications can choose the data read, without knowledge of who is publishing
1
2
3
End Users

Figure 2

Discovery
DDS provides Dynamic Discovery of publishers and subscribers. Dynamic Discovery makes your DDS applications *extensible*. This means the application does not have to know or configure the endpoints for communications because they are automatically discovered by DDS. This dynamic discovery goes even further than discovering endpoints. DDS will discover if the endpoint is publishing data, subscribing to data, or both. It will discover the type of data being published or subscribed to. It will also discover the publisher's offered communication characteristics and the subscriber's requested communications characteristics. All of these attributes are taken into consideration during the dynamic discovery and matching of DDS participants.

DDS participants can be on the same machine or across a network: the application uses the same DDS API for communications. Because there is no need to know or configure IP addresses, or take into account the differences in machine architectures, adding an additional communication participant on any operating system or hardware platform becomes an easy, almost trivial, task.

Strong Type Safety
DDS provides strong type safety by requiring the application to specify the types of data used for communications. The application then reads and writes typed data. For example, a C application will call write() with a specific structure type, not a void pointer.

```
/* Declare a variable with an application-defined data type */
StringMsg    myStringMsg;
/* Call write() with the typed variable, no casting necessary */
StringMsgDataWriter_write ( dw, &myStringMsg, DDS_HANDLE_NIL );
```

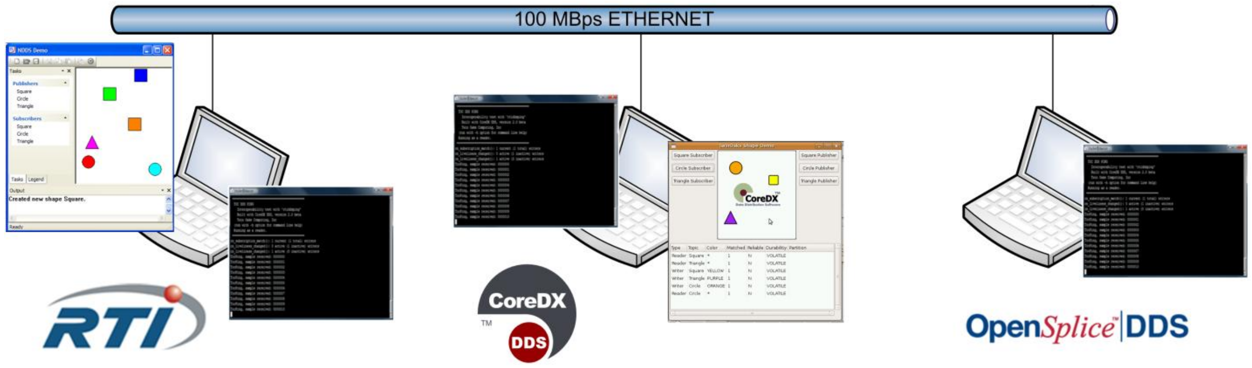
This allows for more errors to be caught early in the development cycle, reducing development costs.
The application defines the data types that will be used for communications using a platform independent language. DDS implementations provide a tool to compile this data type definition into a natural, programming language specific, data type. This
publishing data and subscribing to data over the DDS network.

QoS Policies

DDS provides a rich set of Quality of Service (QoS) policies to tailor the behavior of communications. These QoS policies can be used individually or together to affect a variety of communications aspects, including reliability, performance, persistence of data, and amount of system resources used. These QoS policies set DDS apart from all other communication middleware solutions. The breadth and depth of the configuration available with these QoS policies allow DDS to be a superior choice for communications in a large variety of industries and architectures.

Interoperability

DDS Interoperability is the ability of DDS implementations from different vendors to communicate. The Real-Time Publish Subscribe (RTPS) protocol defines the standardized wire protocol for DDS and is what allows interoperability on the wire between different implementations of DDS.



Architects and system designers in charge of large software systems understand the importance of interoperability, and how difficult it is to obtain. Each DDS implementation offers its own strengths and advantages. Large software systems contain many subsystems and components, and frequently, each of these parts has different networking requirements. When you are writing your own communications software, you can customize it to meet the requirements of each subsystem. Until now, this has not been possible with commercial communication middleware products. However, with today's interoperable DDS implementations, system architects have the flexibility they require to select the DDS vendor that best meets the requirements of each part of their system, and will maintain a flexible, *interoperable* architecture.

Not all DDS implementations are interoperable. There are currently three vendors active in interoperability testing and demonstration: Twin Oaks Computing, RTI, and PrismTech. These vendors are active at the OMG Technical Meetings and regularly test and demonstrate their interoperable DDS products. [http://www.omg.org/news/releases/pr2009/03-25-09.htm]

Performance

DDS is a low-latency communication architecture, and while different implementations of DDS will have different performance characteristics, in general all implementations will be high-performance. This is because the DDS and RTPS specifications are written to satisfy the requirements of real-time and near real-time systems.

These specifications contain instructions to keep data copies to a minimum within the DDS middleware. They also specify a compact data encoding on the wire, light-weight notification mechanisms, and the ability for the application to specify resources limits: allowing DDS to pre-allocate memory and reduce the number of memory allocations during run-time. All of these characteristics result in DDS implementations that are in general efficient, low-overhead, and high-performance.

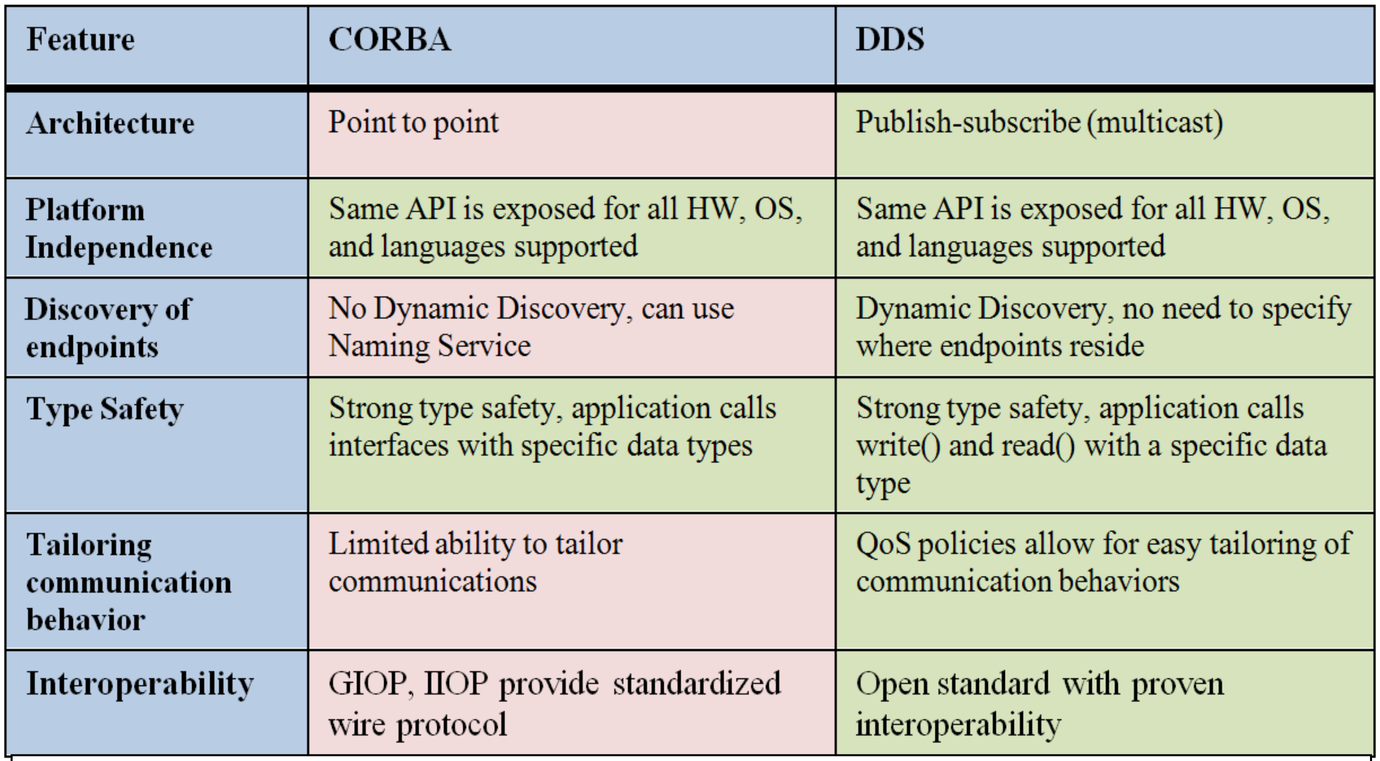
Feature	Standard Socket API	DDS
Architecture	TCP: Point to point UDP: Point to point, broadcast or multicast	Publish-subscribe (multicast)
Platform Independence	Additional code must be written for each HW, OS, and language	Same API is exposed for all HW, OS, and languages supported
Discovery of endpoints	IP addresses, port numbers are hardcoded	Dynamic Discovery, no need to specify where endpoints reside
Type Safety	No type safety, application must convert a stream of bytes to the correct data type	Strong type safety, application calls write() and read() with a specific data type
Tailoring communication behavior	Custom code must be written to tailor behavior	QoS policies allow for easy tailoring of communication behaviors
Interoperability	Not available	Open standard with proven interoperability

DDS Compared to Other Middleware Technologies

DDS vs. Sockets API

The Sockets API has been around for decades, and it continues to be used in virtually all industries requiring data communications. Many developers consider the Sockets API to be the clear choice to meet strict performance requirements or to perform communications on specialized hardware and operating systems. Especially in embedded environments, many other communication middleware technologies simply do not fit. With very constrained memory and disk resources, and often with specialized processors and networking hardware, writing customized code using sockets has been the preferred solution to embedded communications. DDS provides low-latency, high-throughput data communications, similar to the performance that can be achieved with raw Sockets. And, with small footprint DDS implementations, DDS can not only run in resource constrained environments, it can also be easily ported to run on and use custom hardware and operating systems.

The architecture between the TCP Sockets API and DDS is fundamentally different: TCP Sockets are connection oriented, a point-to-point, client-server architecture. This requires that clients and servers know each other's location in order to connect. This is vastly different from the publish-subscribe architecture of DDS.
With DDS, the development of distributed applications is simplified. DDS provides a common API regardless of operating system or hardware architecture. DDS handles the discovery and management of remote endpoints. DDS handles the details of communication behaviors like filtering, reliability, durability, saving historical data, detecting deadline misses and changes in liveliness, and many more. These are all features that do not need to be coded, they are accessed through the standard DDS API. This leaves you more time to focus on the functional requirements of your applications.



Feature	CORBA	DDS
Architecture	Point to point	Publish-subscribe (multicast)
Platform Independence	Same API is exposed for all HW, OS, and languages supported	Same API is exposed for all HW, OS, and languages supported
Discovery of endpoints	No Dynamic Discovery, can use Naming Service	Dynamic Discovery, no need to specify where endpoints reside
Type Safety	Strong type safety, application calls interfaces with specific data types	Strong type safety, application calls write() and read() with a specific data type
Tailoring communication behavior	Limited ability to tailor communications	QoS policies allow for easy tailoring of communication behaviors
Interoperability	GIOP, IIOP provide standardized wire protocol	Open standard with proven interoperability

The above table summarizes the similarities and differences between DDS and CORBA

DDS vs. CORBA
The CORBA and DDS technologies share the same roots as an open standard managed by the OMG. Probably because of this, there are some similarities, especially in the strong type safety provided by both technologies. Although both technologies contain standards for an interoperable wire protocol, DDS vendors have shown a unique cooperation in testing and demonstrating inter-product interoperability.

CORBA is another client-server middleware, based on remote procedure calls, or “remote invocations” in CORBA terminology. CORBA provides a layer of abstraction using an Object Request Broker (ORB) to manage the connections. Clients and Servers need to know about each other directly or use a Naming Service.

Feature	JMS	DDS
Architecture	Publish subscribe	Publish subscribe (multicast)
Platform Independence	Same API is exposed for all HW, OS, and languages supported	Same API is exposed for all HW, OS, and languages supported
Discovery of endpoints	JNDI and JMS servers must be specified and configured	Dynamic Discovery, no need to specify where endpoints reside
Type Safety	Generic Objects and XML are not type safe	Strong type safety, application calls write() and read() with a specific data type
Tailoring communication behavior	Limited ability to tailor communications	QoS policies allow for easy tailoring of communication behaviors
Interoperability	None	Open standard with proven interoperability

The above table summarizes the similarities and differences between DDS and JMS.

The Java Messaging Service (JMS) and DDS are both publish-subscribe middleware technologies. This provides some common ground for comparing the two technologies, but there are also some significant differences.

JMS uses a server for communications that must be configured with the queues or topics that will be used. While publishers and subscribers themselves are loosely coupled, each application participating in JMS communications must connect to a JMS server. The JMS server handles the details of connecting publishers and subscribers. In contrast, DDS does not require any servers or daemons: DDS handles discovery and management of endpoints in each DDS application. Using DDS eliminates the need for a server, and thus the potential single point of failure, and reduces the complexity of the deployed network.

DDS offers some significant benefits that are not available from JMS offerings, including the dynamic discovery of endpoints, no server or daemon process required; the ability to tailor communication behaviors of individual Data Readers and Data Writers through Quality of Service policies; advanced time and content based filtering; and proven interoperability between different vendor implementations.

DDS vs. Service Oriented Architecture (SOA)

Service Oriented Architecture is an architectural style implemented by many technologies such as WS (Web Services), JMS, CORBA, and DDS. DDS distinguishes itself from other SOA publish-subscribe technologies such as JMS and WS Notification by its robust Quality of Service policies including complex time and content based filters. DDS allows application developers to configure the urgency, importance, reliability, and persistence of each piece of data to be communicated. DDS makes an excellent interoperable publish subscribe SOA component.

Feature	Web Services	DDS
Architecture	Message oriented Middleware	Publish subscribe (multicast)
Platform Independence	Same API is exposed for all HW, OS, and languages supported. Browser clients can run anywhere	Same API is exposed for all HW, OS, and languages supported
Discovery of endpoints	Web server must be specified and configured	Dynamic Discovery, no need to specify where endpoints reside
Type Safety	Use of HTTP and XML does not provide strong type safety	Strong type safety, application calls write() and read() with a specific data type
Tailoring communication behavior	Limited ability to tailor communications	QoS policies allow for easy tailoring of communication behaviors
Interoperability	Some interoperability with multiple protocols used: SOAP, REST, WSDL	Open standard with proven interoperability

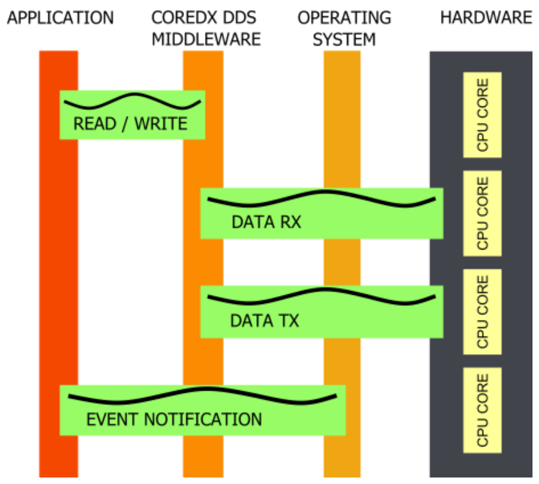
Why is CoreDX DDS different?

CoreDX DDS, an implementation of the DDS and RTPS standards, provides a number of significant technical benefits over leading commercial and open source DDS implementations.

Small Footprint

CoreDX DDS is a full-featured DDS implementation that comes in a surprisingly small package. The entire 'C' library, containing full interoperability with the RTPS wire protocol and support for all the standard, and a some additional QoS policies, is a mere 500 KB. That's Kilobytes, not Megabytes! Our CoreDX DDS Ping application (text-based application built to test interoperability over lots of QoS policies) is 250 KB.

While CoreDX DDS can run on enterprise-class servers, it does not require expensive compute platforms. CoreDX DDS makes a DDS application run on standard, Intel-based desktops and laptops, Single Board Computers (SBCs) with Intel or PPC processors, and even smaller form factor computers using ARM processors. Applications built with the standard CoreDX DDS product can run on machines with less than 640 KB of memory.



Advanced Multicore Support

With the growing prevalence of multi-core computer systems, software architects are presented with many opportunities for performance improvements; however, parallel-programming environments raise significant complexity challenges. Programming languages and Operating Systems provide some tools to help reduce the complexity of parallel programming; but these fundamental tools are complex in themselves.

The CoreDX DDS middleware simplifies the task of putting additional cores to work. The engineers at Twin Oaks Computing have worked with multi-threaded and parallel programming environments for over a decade. This expertise has been incorporated into the advanced data pipeline architecture in CoreDX DDS™. By internally pipelining the flow of data, CoreDX DDS™ can employ multiple processing cores simultaneously. This automatic distribution of work across multiple cores is easy to exploit in application code because of the flexibility of the CoreDX DDS™ API.

With these tools, developers can write application software with a single thread of control, and the CoreDX DDS™ middleware will distribute the communications work across multiple cores. This simplifies the application code while employing sophisticated parallel programming technologies.

Low Line of Code Count

The number of software lines of code (SLOC) in a software product is not something every customer considers. Of course, SLOC counts are important to systems with safety critical requirements. These systems must certify each line of code, and 3rd party software packages like communication middleware are not exempt from this certification process. This adds an additional cost to every line of code in the system. However, SLOC counts also have implications outside of the safety critical markets.

Every line of code in a software product has a cost associated with it, and not just the cost of certification for safety critical applications. Twin Oaks Computing not only understands this; this idea is the foundation for our software development processes. Each line of code developed for CoreDX DDS must be tested and maintained over the life of our product. Larger software baselines will have greater test and maintenance expenses. Each line of code has the potential for increasing the number of instructions that must be executed, and degrading the overall performance of communications. In addition, each line of code has the potential for a programming error, or bug, and more lines of code make it difficult to track down and identify such errors.

These are fundamental concepts in any software development project, and truly experienced software engineers understand the importance of writing code that is well thought out and compact. Every engineer at Twin Oaks Computing understands these concepts and follows a specific software process for making changes to the CoreDX DDS baseline. Each software addition and modification is carefully analyzed for its SLOC and performance impact and benefit to the overall product. If we are not happy with the end result, we go back to the drawing board for another solution. These fundamental software engineering concepts and complementing processes ensure the CoreDX DDS baseline maintains it status as the World Leading Small Footprint DDS Implementation.

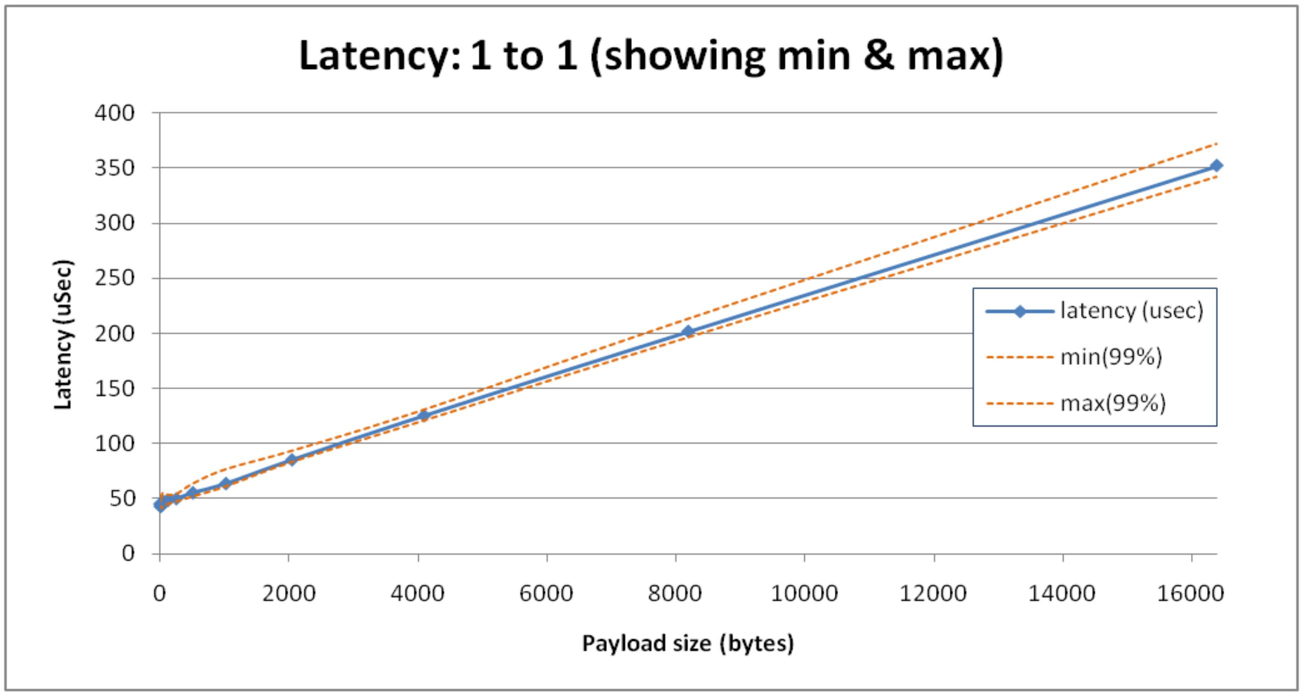
The complete CoreDX DDS baseline includes fewer than 35,000 SLOC for the Standard Edition. Twin Oaks Computing also has a Safety Critical Baseline for the CoreDX DDS product, which has fewer than 13,000 SLOC. This new Safety Critical Baseline includes all the QoS policies and features of the standard CoreDX DDS baseline. In fact, the real difference between the two baselines is the discovery process: the Safety Critical Baseline contains a modified (less dynamic) discovery process.

Performance

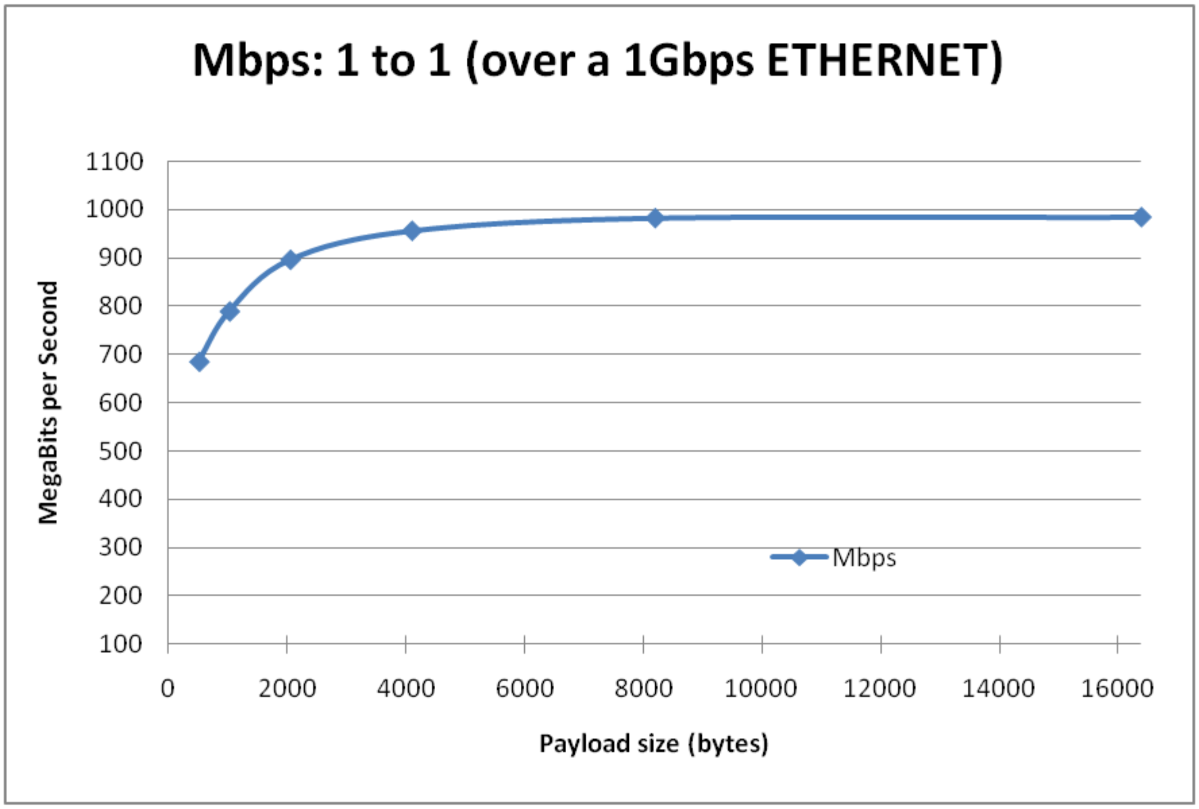
The performance of most DDS implementations will be good – very good when compared with JMS and other XML-based communication middleware technologies. The performance of CoreDX DDS goes further. For example, on a 1Gb network, CoreDX DDS can provide throughputs over 900 Mbps, and latencies around 60 microseconds.

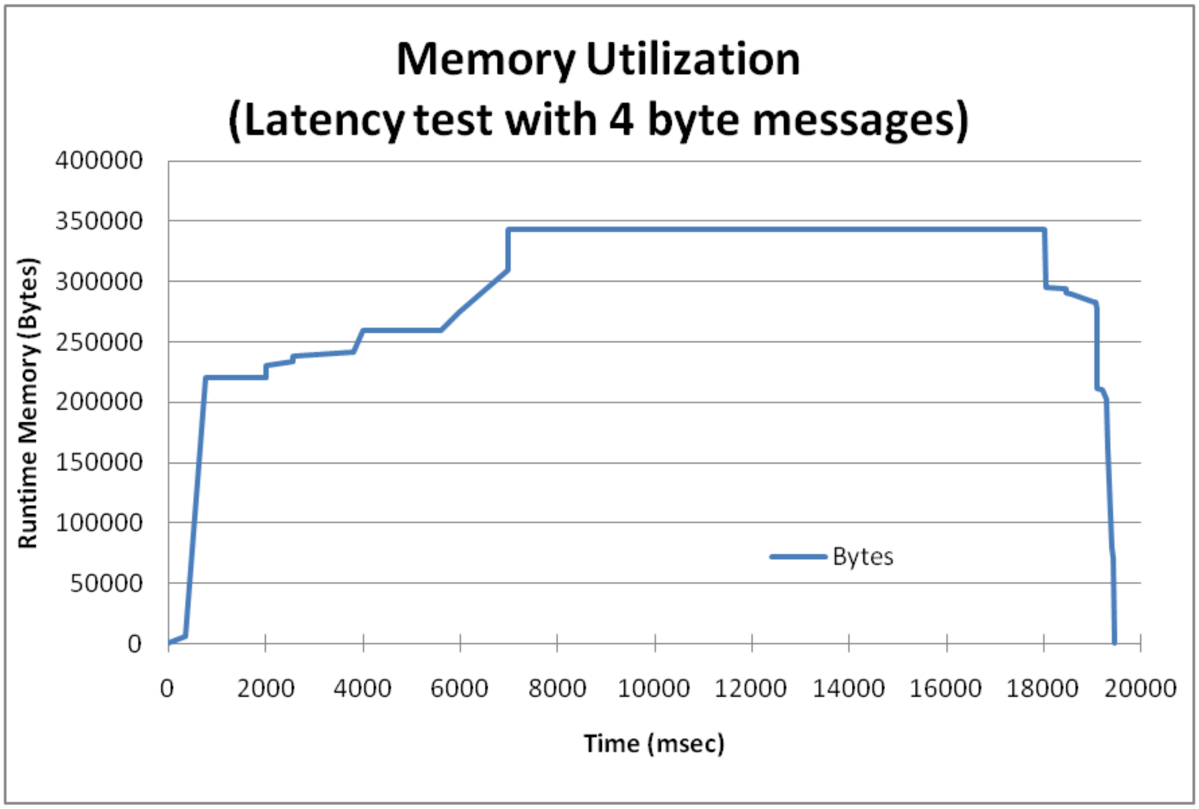
The performance that can be achieved with any communication middleware is going to depend on the network and machine configurations: the switch hardware, network interface card (NIC), operating system, and network device driver all impact the baseline throughput and latencies that can be achieved on a system.

The application developer has control over the performance achieved by adjusting how CoreDX DDS is configured and used.



TWINOAKS COMPUTING INC.
PRACTICAL MIDDLEWARE EXPERTISE





TWINOAKS COMPUTING INC. PRACTICAL MIDDLEWARE EXPERTISE

CoreDX DDS provides a number of configuration options that will affect the performance of the system. First, the data types employed will affect performance. In general, data types that result in smaller message sizes will provide lower latencies, while data types that result in larger message sizes will provide better throughput. In addition, many DDS QoS policies will affect performance, including Reliability, History, Resource Limits, Latency Budget, and Durability. In general, the best performance can be achieved by setting Resource Limits and using data types that are fixed sizes (allowing CoreDX DDS to reduce the amount of memory allocation performed during operations by pre-allocating the necessary resources). In addition, throughput can be improved by using larger message sizes and longer Latency Budgets. By contrast, latency results can be optimized by using smaller message sizes and no Latency Budgets.

Dynamic Types

A feature exclusive to CoreDX DDS is support for Dynamic Type Technology. This innovative new technology eases system integration challenges, and enables bridging DDS data between disparate systems in a flexible and dynamic environment. This technology enables DataReaders to dynamically, at run-time, determine the topic data types. Through Dynamic Type introspection, the subscribing application can explore the data type and access data fields. Dynamic Type Technology is a critical component for the deployment of useful DDS bridging solutions. Without this technology, DDS bridges must be maintained and upgraded in lock-step with the systems they support. Dynamic Types offer a flexible solution that lowers Total Cost of Ownership.

Conclusion

Using DDS for data communications between distributed applications provides a number of benefits over other traditional data communication solutions. Benefits of DDS include Dynamic Discovery, strong Type Safety, a wide variety of configuration options in the QoS policies, Interoperability, and Performance.

While the DDS technology has been available for nearly two decades, until recently these implementations required resources not available to embedded application designers. Today, all the benefits of DDS are available in a significantly smaller footprint, the perfect solution for not only embedded environments, but also safety critical applications that must certify every line of code.

About Twin Oaks Computing

With corporate headquarters located in Castle Rock, Colorado, USA, Twin Oaks Computing is a company dedicated to developing and delivering quality software solutions. We leverage our technical experience and abilities to provide innovative and useful services in the domain of data communications. Founded in 2005, Twin Oaks Computing, Inc delivered the first version of CoreDX DDS in 2008. The next two years saw deliveries to over 100 customers around the world. We continue to provide world class support to these customers while ever expanding.

Contact

Twin Oaks Computing, Inc.
(720) 733-7906
+33 (0)9 62 23 72 20
755 Maleta Lane
Suite 203
Castle Rock, CO. 80108
www.twinoakscomputing.com