OpenSplice

Version 6.x

Evaluation & Benchmarking Guide

PrismTech

1 Introduction

One of the key differentiators of OpenSplice Enterprise is that it provides a user with the ability to choose exactly how to deploy Data Distribution Service (DDS) applications, *i.e.* there are *different DDS system architecture deployment modes* and also *different networking service protocols*. This allows a user to maximize both intra-nodal and inter-nodal performance based on requirements specific to their own use case. When evaluating OpenSplice Enterprise it is very important to understand all of these features and benefits to ensure that the most appropriate combination is evaluated against your specific performance criteria. Once the performance figures have been observed the choice is usually clear.

Every customer use case and set of requirements is different, so let us briefly guide you through how to best deploy OpenSplice Enterprise so that it meets and exceeds your expectations. Here we explain how easy it is to get started with OpenSplice Enterprise and observe the excellent performance and scalability it provides. OpenSplice Enterprise is even shipped with dedicated performance tests that the user can build and run easily.

Note that this *Guide* serves only as an introduction and does not replace the full OpenSplice Enterprise reference and user guides.

2 OpenSplice Enterprise Basics

OpenSplice Enterprise is configured using an XML configuration file. In this file, the user specifies the architectural model and OpenSplice Enterprise services that are to run when the DDS infrastructure is started.

The `OSPL_URI` environment variable refers to the specific XML configuration file that is used for the current deployment. The default value refers to the `ospl.xml` file located in the `etc/config` directory of the OpenSplice Enterprise installation. The installation directory itself can be referred to by the `OSPL_HOME` environment variable. Please see section *6.1* on page 11 for details of how to set up the OpenSplice Enterprise environment.

A number of other sample configuration files that can be used when benchmarking OpenSplice Enterprise are also provided in the `etc/config` directory.

The `OSPL_URI` variable is of the form:

Linux

```
OSPL_URI=file://$OSPL_HOME/etc/config/ospl.xml
```

WIN

```
OSPL_URI=file://%OSPL_HOME%\etc\config\ospl.xml
```

Please refer to the `OpenSplice_Deployment.pdf` for more details of the `OSPL_URI` variable, but now let's see what aspects of the OpenSplice deployment are controlled by this file.

3 OpenSplice Architectural Modes

*OpenSplice Enterprise provides two main architectural modes. These are the **Single Process** deployment mode which provides a **Standalone** architecture, and, unique to OpenSplice, the **Shared Memory** deployment mode which provides a **Federated** architecture.*

3.1 The Single Process or Standalone deployment

Features of this mode are:

- Simplest to run and get started with
- Each DDS application process contains the entire DDS infrastructure
- Uses in-process heap memory for the DDS database
- OpenSplice Enterprise services run as threads within the application process
- When there are multiple DDS application processes on a single machine, the communication between them must be done *via* a networking service. This induces an overhead so performance in this scenario is not optimal

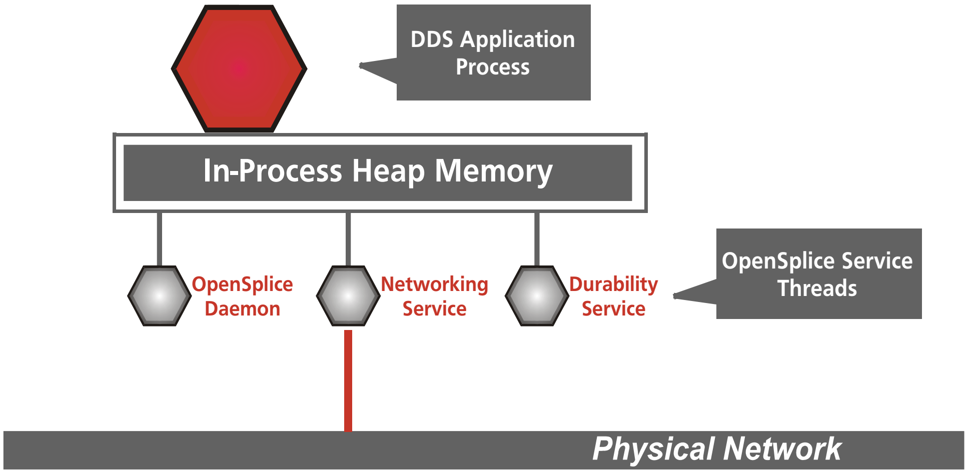

Figure 1 Single Process or Standalone deployment

3.2 The Shared Memory or Federated deployment

Features of this mode are:

- The DDS infrastructure is started once per machine
- Uses shared memory for the DDS database

- Each DDS application process interfaces with the shared memory rather than creating the DDS infrastructure itself
- Allows the data to be physically present only once on any machine
- Reading and writing directly to locally-mapped memory is far more efficient than having to actually move the data *via* a networking service, allowing for improved performance and scalability
- OpenSplice Enterprise services are able arbitrate over all of the DDS data on the node, and so can make smart decisions with respect to data delivery so that priority QoS values (for example) are respected. That is not possible when there are multiple standalone deployments on a machine

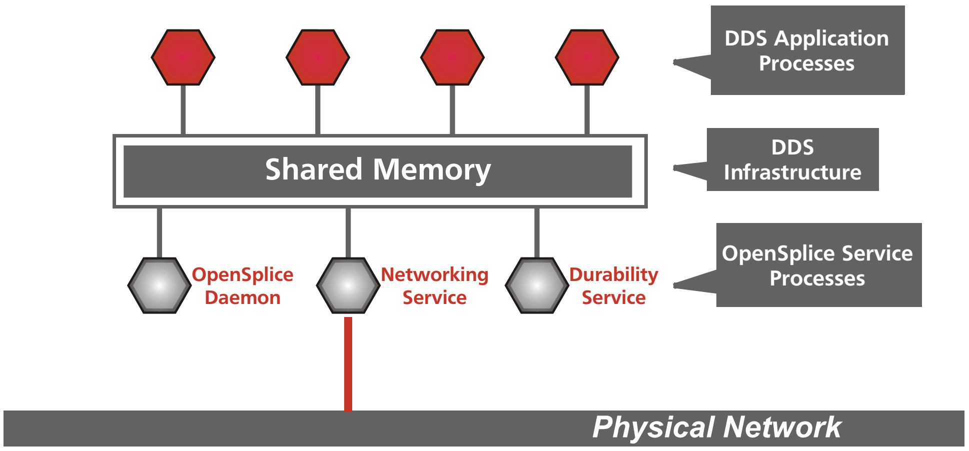

Figure 2 Shared Memory or Federated deployment

When there are multiple DDS applications running on a single computing node, the use of OpenSplice's unique Shared Memory architecture can provide greater performance, smaller footprint and better scalability than other DDS deployment options.

3.3 How to select the Architectural Mode

- For a ***Single Process*** deployment, set the `OSPL_URI` variable to refer to a single process (sp) xml file such as `ospl_sp_ddsi.xml` or `ospl_sp_nativeRT.xml`. Note that a networking service (such as ddsi or nativeRT) is required for two DDS application processes to communicate. See the next section for more details on networking options.

A single process deployment is enabled when the `Domain` section of the XML configuration contains a `<SingleProcess> TRUE` attribute.

- For a ***Shared Memory*** deployment, set the `OSPL_URI` variable to refer to a shared memory (`shmem`) xml file such as `ospl_shmem_no_network.xml`, `ospl_shmem_ddsi.xml`, or `ospl_shmem_nativeRT.xml`. A networking service is not required for two DDS application processes to communicate on the same node.

A shared memory deployment is enabled when the `Domain` section of the XML configuration does not contain a `<SingleProcess> TRUE` attribute but does contain a `<Database>` attribute.

Note that by default the `OSPL_URI` environment variable refers to a *Single Process* configuration, so to see the extra performance and scalability benefits of OpenSplice DDS's Shared Memory architecture it is necessary to switch from the default.

4 OpenSplice Networking Options

OpenSplice Enterprise provides several networking options for the delivery of DDS data between nodes. The networking service selection is largely transparent to the user—the difference is observed in the CPU consumption, networking load, and ultimately how fast and efficiently the data is delivered between nodes. The most applicable service is dependent on the requirements of the use case.

OpenSplice DDSI is the industry standard protocol providing vendor interoperability that operates using a typed ‘pull’ style model.

OpenSplice RTNetworking is an alternative to the DDSI wire protocol. RTNetworking uses a type-less ‘push’ style model in contrast to DDSI and is often the more performant, scalable option. RTNetworking also offers prioritization of network traffic *via* ‘channels’, partitioning to separate data flows and optional compression for low-bandwidth environments. ***OpenSplice SecureRTNetworking*** provides these features together with encryption and access control.

OpenSplice DDSI2E is the “enhanced” version of the interoperable service. DDSI2E offers the benefits of the DDSI protocol (such as its automatic unicast delivery in the case of there being a single subscribing endpoint), together with some of the performance features of the RTNetworking service such as channels, partitioning and encryption.

4.1 How to select the Networking Protocol

As with the architectural deployment choice, the selection of the networking service is described by the XML configuration file. Note that this choice is independent of and orthogonal to the architectural deployment: you can have single process or shared memory with any of the networking service protocols.

- To run with a ***DDSI*** service, set the `OSPL_URI` variable to refer to a DDSI xml file such as `ospl_sp_ddsi.xml` or `ospl_shmem_ddsi.xml`.
- To run with an ***RTNetworking*** service, set the `OSPL_URI` variable to refer to an RTNetworking xml file such as `ospl_sp_nativeRT.xml` or `ospl_shmem_nativeRT.xml`.
- To run with a ***SecureRTNetworking*** service, set the `OSPL_URI` variable to refer to the `ospl_shem_secure_nativeRT.xml` SecureRTNetworking xml file.
- To run with a ***DDSI2E*** deployment, set the `OSPL_URI` variable to refer to a DDSI2E xml file such as `ospl_sp_ddsi2e.xml` or `ospl_shmem_ddsi2e.xml`.

Note that by default, the `OSPL_URI` environment variable refers to a *DDSI* configuration, so to see the extra performance and scalability benefits of OpenSplice DDS’s RTNetworking or DDSI2E it is necessary to switch from the default.

5 Benchmarking OpenSplice: Decision Trees

DDS provides many functional benefits that set it apart from other middleware technologies, but users often still have specific performance requirements for ***latency****,* ***throughput****,* ***CPU*** *and* ***network utilization****. OpenSplice Enterprise provides the functional benefits of the technology whilst remaining committed to excellent performance.*

The flowcharts overleaf show the decision criteria that may be applied in order to decide on the most appropriate test case (*Figure 3*), architectural mode (*Figure 4*) and networking protocol (*Figure 5*) options for your specific use case and requirements.

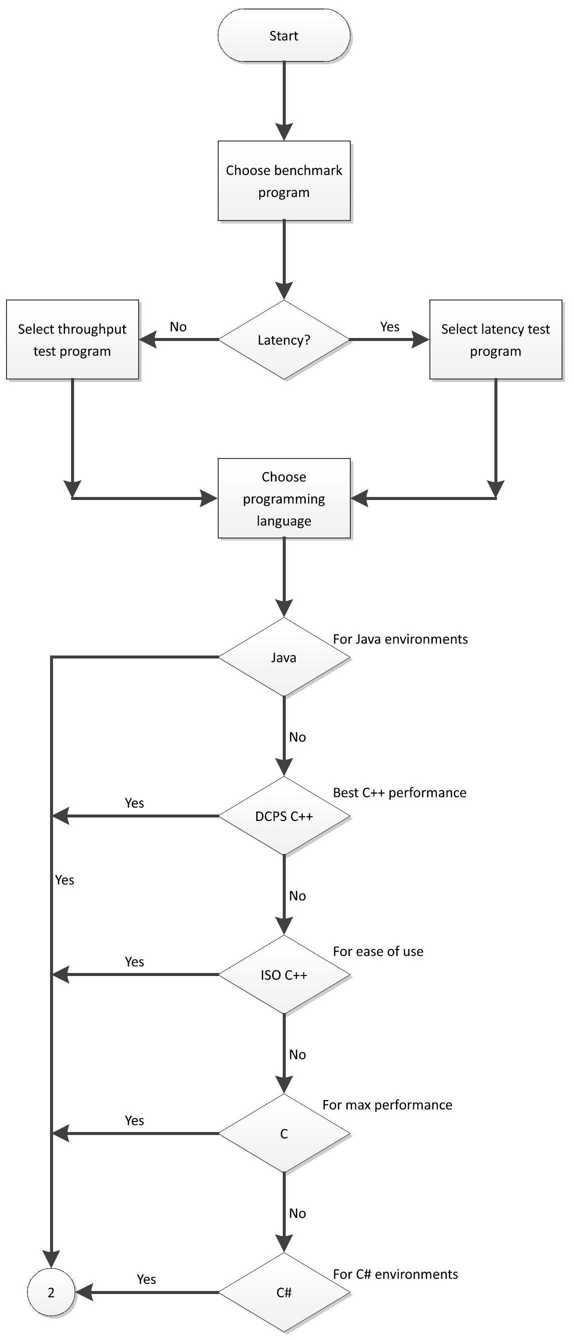

Figure 3 Selecting a specific performance test and programming language

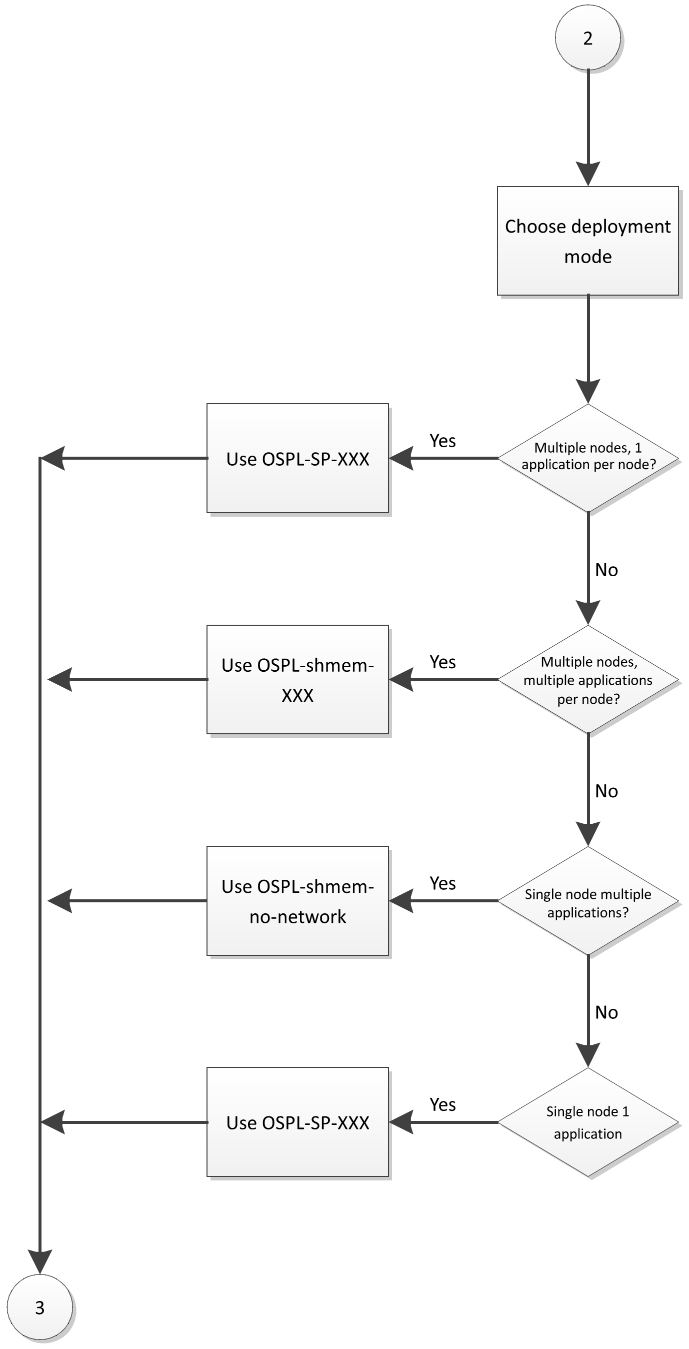
2
Choose deployment mode
Multiple nodes, 1 application per node?
Yes
Use OSPL-SP-XXX
No
Multiple nodes, multiple applications per node?
Yes
Use OSPL-shmem-XXX
No
Single node multiple applications?
Yes
Use OSPL-shmem-no-network
No
Single node 1 application
Yes
Use OSPL-SP-XXX
3

Figure 4 Selecting the architectural deployment mode

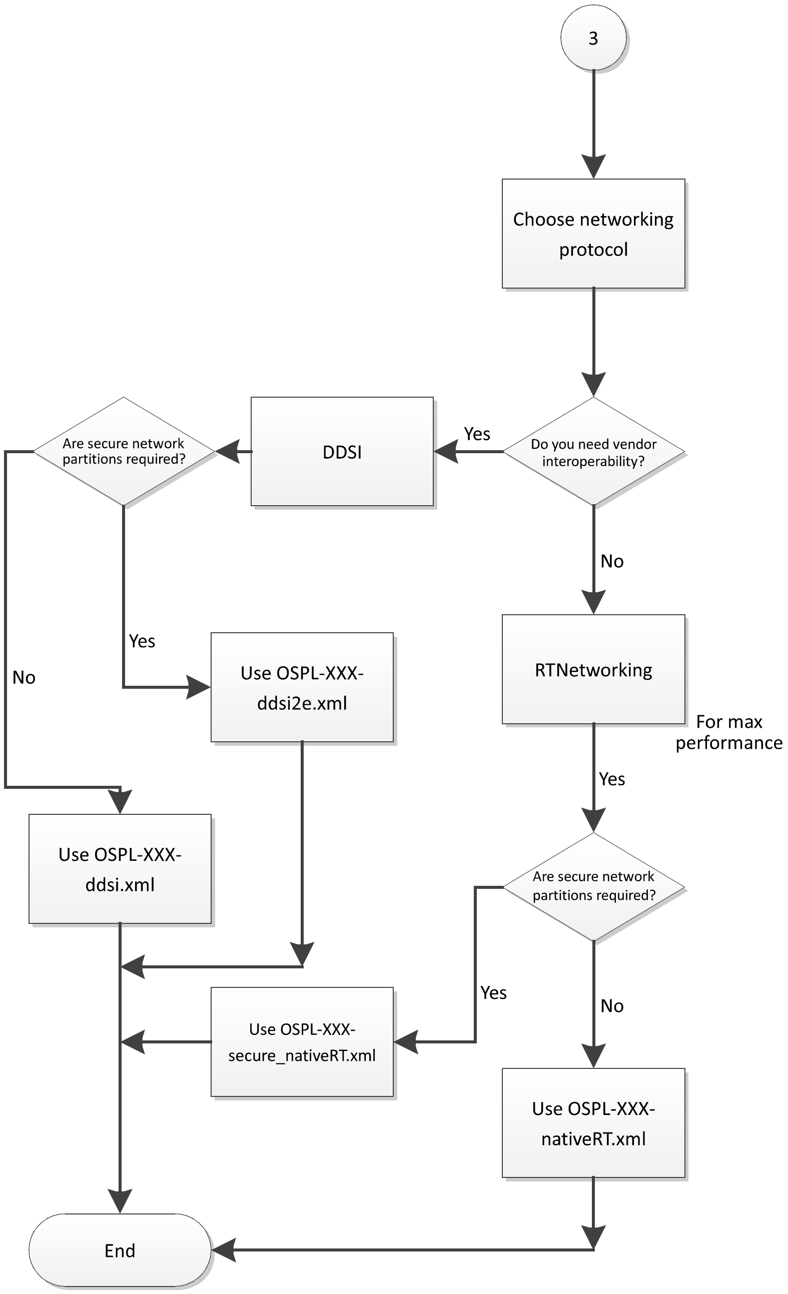
3
Choose networking protocol
Do you need vendor interoperability?
Yes
DDSI
Are secure network partitions required?
Yes
Use OSPL-XXX-ddsi2e.xml
No
Use OSPL-XXX-ddsi.xml
No
RTNetworking
For max performance
Yes
Are secure network partitions required?
Yes
Use OSPL-XXX-secure_nativeRT.xml
No
Use OSPL-XXX-nativeRT.xml
End

Figure 5 Selecting the network protocol options

6 How to run OpenSplice Enterprise

6.1 The OpenSplice Enterprise Environment

A `release` file is provided with the OpenSplice Enterprise installation which contains the environment variables that are required.

Create an OpenSplice Enterprise environment as follows:

Linux Open a shell and source the `release.com` file from the OpenSplice Enterprise installation directory.

WIN Open a Windows Command prompt and run the `release.bat` file in the OpenSplice Enterprise installation directory.

WIN Alternatively, use the 'OpenSplice DDS Command Prompt' that can be accessed from the Windows Start menu (this will implicitly run `release.bat`).

Next, set the `OSPL_URI` variable to refer to the OpenSplice Enterprise configuration that is required (see section 3.3, *How to select the Architectural Mode*, on page 4).

6.2 Running Single Process and Shared Memory Modes

With an `OSPL_URI` variable referring to a ***Single Process*** deployment, you just need to start the DDS application process. The `create_participant()` operation, which is the entry into the DDS Domain, will create the entire DDS infrastructure within the application process and the services will be started as threads.

With an `OSPL_URI` variable referring to a ***Shared Memory*** deployment, it is necessary to start the DDS infrastructure before starting your DDS application processes. That is done by using the `ospl` utility tool:

```
ospl start

# now run the DDS application processes as normal

ospl stop
```

7 Performance Tests and Examples

To make the evaluation process as easy as possible, OpenSplice Enterprise is shipped with dedicated performance tests that can be used to measure latency and throughput. The tests are simple and clear, allowing the user to obtain performance results easily.

Performance tests are currently provided for the C++ and ISO C++ APIs.

7.1 Round Trip Latency Performance

The ***latency*** of a DDS implementation is an expression of how fast data can be delivered between two DDS applications. *Round-trip latency* is the time taken for an individual DDS data sample to be delivered from Application A to Application B and back again, so importantly it includes metrics for both data delivery and reception.

To build and run the round-trip performance test, for example for the ISO C++ API:

Linux

```
# In an OpenSplice Enterprise environment:
cd $OSPL_HOME/examples/dcps/RoundTrip/isocpp
make

cd $OSPL_HOME/examples/dcps/RoundTrip/isocpp
# If using shared memory do “ospl start”
./pong
# If using shared memory do “ospl stop”

# In another OpenSplice Enterprise environment:
cd $OSPL_HOME/examples/dcps/RoundTrip/isocpp
# If using shared memory do “ospl start”
./ping 20 100
# If using shared memory do “ospl stop”
```

WIN

```
# Load the OpenSplice DDS examples project solution in to Visual
Studio and build the required projects

# In an OpenSplice Enterprise environment:
cd %OSPL_HOME%\examples\dcps\RoundTrip\isocpp
# If using shared memory do “ospl start”
pong.exe
# If using shared memory do “ospl stop”

# In another OpenSplice Enterprise environment:
cd %OSPL_HOME%\examples\dcps\RoundTrip\isocpp
# If using shared memory do “ospl start”
ping.exe 20 100
# If using shared memory do “ospl stop”
```

The `ping` application will report the roundtrip time taken to send DDS data samples back and forth between the applications. The test utilizes the ReliabilityQoS set to `RELIABLE` by default in order to show the maximal performance whilst maintaining the guaranteed delivery of DDS samples. See the `README` file for the test for further details.

The lowest roundtrip latency may be achieved by tuning the test parameters appropriately.

Note that the default `OSPL_URI` value refers to a ***Single Process*** deployment with ***DDSI*** networking.

- To observe the best performance *within* a node it is suggested that you use a ***Shared Memory*** configuration.
- To observe the best performance *between* nodes it is suggested that you use an ***RTNetworking*** service configuration.

7.2 Throughput Performance

The ***throughput*** of a DDS implementation is an expression of the rate of data delivery through the DDS system. Measured in bits per second, it describes the ability of the DDS implementation to effectively deliver DDS data without data loss.

To build and run the throughput performance test, for example for the ISO C++ API:

Linux

```
# In an OpenSplice Enterprise environment:
cd $OSPL_HOME/examples/dcps/Throughput/isocpp
make

cd $OSPL_HOME/examples/dcps/Throughput/isocpp
# If using shared memory do “ospl start”
./publisher
# If using shared memory do “ospl stop”

# In another In an OpenSplice Enterprise environment:
cd $OSPL_HOME/examples/dcps/Throughput/isocpp
# If using shared memory do “ospl start”
./subscriber
# If using shared memory do “ospl stop”
```

WIN

```
# Load the OpenSplice DDS examples project solution in to Visual
Studio and build the required projects

# In an OpenSplice Enterprise environment:
cd %OSPL_HOME%\examples\dcps\Throughput\isocpp
# If using shared memory do “ospl start”
publisher.exe
# If using shared memory do “ospl stop”

# In another OpenSplice Enterprise environment:
cd %OSPL_HOME%\examples\dcps\Throughput\isocpp
# If using shared memory do “ospl start”
subscriber.exe
# If using shared memory do “ospl stop”
```

The `subscriber` application will report the DDS data throughput by default once per second. This and many other aspects of the test can be configured on the command line. The test utilizes the ReliabilityQoS set to `RELIABLE` by default in order to show the maximal performance whilst maintaining the guaranteed delivery of DDS samples. See the `README` file for the test for further details.

The maximum throughput may be achieved by tuning the test parameters appropriately.

Note that the default `OSPL_URI` value refers to a ***Single Process*** deployment with ***DDSI*** networking.

- To observe the best performance *within* a node it is suggested that you use a ***Shared Memory*** configuration.
- To observe the best performance *between* nodes it is suggested that you use an ***RTNetworking*** service configuration.

7.2.1 Achieving Maximum Throughput

Where there is a requirement to support continuous flows or ‘streams’ of data with minimal overhead consider the use of ***OpenSplice Streams***. The ability to deliver potentially millions of samples per second is realized by the Streams feature transparently batching (packing and queuing) the periodic samples.

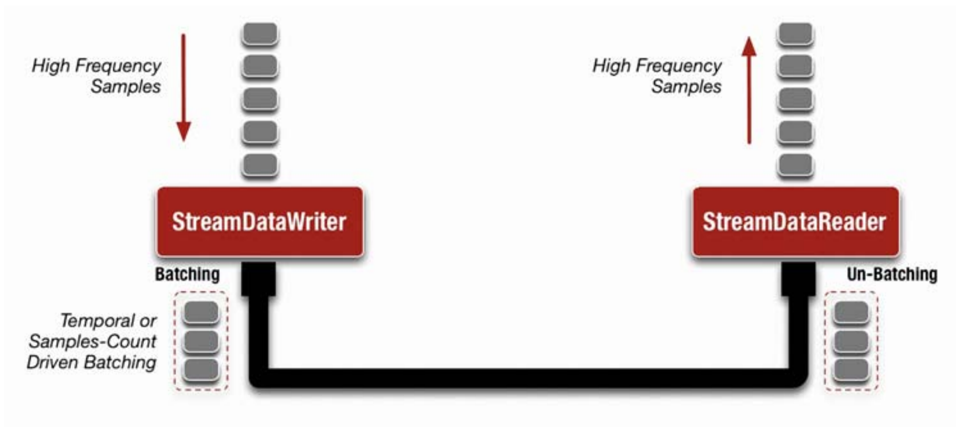

Figure 6 Streams Architecture

The streams performance example is located in the `examples/streams` directory within the installation.

Bibliography

[1] *Data Distribution Service for Real-Time Systems Version 1.2*, OMG
Available specification formal/07-01-01

[2] *The Real-Time Publish-Subscribe Wire Protocol DDS Interoperability Wire Protocol Specification Version 2.1*, OMG
Document Number: formal/2009-01-05

[3] *Extensible and Dynamic Topic Types for DDS Version 1.0*, OMG
Document Number: formal/2012-11-10

Contacts

USA Corporate Headquarters

PrismTech Corporation
400 TradeCenter
Suite 5900
Woburn, MA
01801
USA

Tel: +1 781 569 5819

European Head Office

PrismTech Limited
PrismTech House
5th Avenue Business Park
Gateshead
NE11 0NG
UK

Tel: +44 (0)191 497 9900
Fax: +44 (0)191 497 9901

PrismTech France

28 rue Jean Rostand
91400 Orsay
France

Tel: +33 (1) 69 015354

Web:	*http://www.prismtech.com*
E-mail:	*info@prismtech.com*

Copyright Notice