

RTI Connex DDS

Combined Latency and Throughput Performance Test

Getting Started Guide

Version 5.2.0



Your systems. Working as one.



© 2015 Real-Time Innovations, Inc.
All rights reserved.
Printed in U.S.A. First printing.
August 2015.

Trademarks

Real-Time Innovations, RTI, NDDS, RTI Data Distribution Service, DataBus, Connex, Micro DDS, the RTI logo, 1RTI and the phrase, "Your Systems. Working as one," are registered trademarks, trademarks or service marks of Real-Time Innovations, Inc. All other trademarks belong to their respective owners.

Copy and Use Restrictions

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form (including electronic, mechanical, photocopy, and facsimile) without the prior written permission of Real-Time Innovations, Inc. The software described in this document is furnished under and subject to the RTI software license agreement. The software may be used or copied only under the terms of the license agreement.

Technical Support

Real-Time Innovations, Inc.
232 E. Java Drive
Sunnyvale, CA 94089
Phone: (408) 990-7444
Email: support@rti.com
Website: <https://support.rti.com/>

Contents

1	Supported Platforms.....	1
2	Release Notes	2
2.1	What's New in 5.2.0.....	2
2.1.1	New Platforms	2
2.1.2	Removed Platforms.....	2
2.1.3	New Command-Line Options	3
2.1.4	Added Support for Modern API C++ Implementation.....	3
2.1.5	Added 99.9999 Percentile in Latency Report	3
2.1.6	C++ Implementations (Traditional and Modern API) Now use High Precision Clock to Measure Time.....	3
2.1.7	Removed .ini Configuration File.....	3
2.2	What's Fixed in 5.2.0	3
2.2.1	Degraded Performance when Using Turbo Mode	3
2.2.2	Potential Segmentation Fault due to Invalid Message ID	3
3	Overview	4
4	Download Instructions.....	4
5	Installation Instructions.....	4
5.1	UNIX-Based Systems	4
5.2	Windows Systems.....	5
6	Building the Application	5
7	Starting the Test	6
8	Test Parameters	8
8.1	Spinning vs. Sleeping.....	14
8.2	Send-Queue Size and Queue-Full Behavior	14
8.3	Number of Iterations vs. Latency Count.....	15
8.4	Warming Up	15
8.5	WaitSet Event Count and Delay	15
8.6	How to Measure Latency for a Given Throughput	16
8.7	Auto Tuning and Turbo Mode	17

9	Example Command Lines for Running the Performance Test	17
9.1	1-1, Multicast, Best Latency as a Function of Message Size	17
9.2	1-1, Multicast, Maximum Throughput as a Function of Message Size (with Batching)	18
9.3	1-1, Multicast, Latency vs. Throughput for 200-byte Messages (with Batching)	18
9.4	1-to-1, Multicast, Reliable UDPv4, All Sizes	18
9.5	1-to-1, Unicast, Best-Effort, UDPv4, 1 Size.....	18
9.6	1-to-1, Multicast, Reliable, UDPv4, Batching Enabled	19
9.7	1-to-2, Multicast, Reliable, UDPv4	19
9.8	2-to-1, Multicast, Reliable, UDPv4	19
10	Example Output.....	20
11	Optimizing Your OS for Network Performance.....	20
11.1	Optimizing Linux Systems.....	20
11.2	Optimizing Windows Systems	20

Testing Performance

This document describes how to run a combined latency and throughput test application for *RTI[®] Connex[™] DDS*.

The test measures what is sometimes called "loaded latency"—latency at a given throughput level. It can help you answer questions such as:

- ☐ For a given publishing configuration (e.g., queue size, batching settings), sample size, and subscribing configuration (e.g., queue size, Listener vs. WaitSet) what is the throughput of my network?
- ☐ When my network is heavily loaded, what latency can I expect?
- ☐ For a given configuration, what is the best-case latency with no other traffic on the network?

For examples of specific configurations, see [Example Command Lines for Running the Performance Test \(Section 9\)](#).

1 Supported Platforms

The performance test application provides makefiles/projects for the following architectures. You can also create your own.

C++ on a UNIX-based System	x32	CentOS 6.x (2.6 kernel)	i86Linux2.6gcc4.4.5
		Red Hat Enterprise Linux 4.0 (2.6 kernel)	i86Linux2.6gcc3.4.3
		Red Hat Enterprise Linux 5.0 (2.6 kernel)	i86Linux2.6gcc4.1.1
		Red Hat Enterprise Linux 6.x	i86Linux2.6gcc4.4.5
	x64	CentOS 5.4, 5.5 (2.6 kernel)	x64Linux2.6gcc4.1.2
		CentOS 6.x (2.6 kernel)	x64Linux2.6gcc4.4.5
		OS X 10.8 (Darwin 12)	x64Darwin12clang4.1
		OS X 10.10 (Darwin 14)	x64Darwin14clang6.0
		Red Hat Enterprise Linux 4.0 (2.6 kernel)	x64Linux2.6gcc3.4.5
		Red Hat Enterprise Linux 5.0 (2.6 kernel)	x64Linux2.6gcc4.1.1
		Red Hat Enterprise Linux 6.x (2.6 kernel)	x64Linux2.6gcc4.4.5
		Red Hat Enterprise Linux 5.1, 5.2, 5.4, 5.5 (2.6 kernel)	x64Linux2.6gcc4.1.2
		Ubuntu 12.04 (3.x kernel)	x64Linux3.xgcc4.6.3

C++ on a Windows System	x32	Visual Studio 2008	i86Win32VS2008
		Visual Studio 2010	i86Win32VS2010
		Visual Studio 2012	i86Win32VS2012
		Visual Studio 2013	i86Win32VS2013
	x64	Visual Studio 2008	x64Win32VS2008
		Visual Studio 2010	x64Win32VS2010
		Visual Studio 2012	x64Win32VS2012
		Visual Studio 2013	x64Win32VS2013
C# on a Windows System	x32	Visual Studio 2008	i86Win32VS2008
		Visual Studio 2010	i86Win32VS2010
		Visual Studio 2012	i86Win32VS2012
		Visual Studio 2013	i86Win32VS2013
	x64	Visual Studio 2008	x64Win64VS2008
		Visual Studio 2010	x64Win64VS2010
		Visual Studio 2012	x64Win64VS2012
		Visual Studio 2013	x64Win32VS2013

2 Release Notes

2.1 What's New in 5.2.0

2.1.1 New Platforms

This release adds support for these platforms:

C++ on a UNIX-based System	x32	CentOS 6.x (2.6 kernel)	i86Linux2.6gcc4.4.5
		Red Hat Enterprise Linux 6.x	i86Linux2.6gcc4.4.5
	x64	CentOS 6.x (2.6 kernel)	x64Linux2.6gcc4.4.5
		OS X 10.8 (Darwin 12)	x64Darwin12clang4.1
		OS X 10.10 (Darwin 14)	x64Darwin14clang6.0
		Red Hat Enterprise Linux 6.x (2.6 kernel)	x64Linux2.6gcc4.4.5
		Ubuntu 12.04 (3.x kernel)	x64Linux3.xgcc4.6.3
C++ on a Windows System	x32	Visual Studio 2013	i86Win32VS2013
	x64	Visual Studio 2013	x64Win32VS2013
C# on a Windows System	x32	Visual Studio 2013	i86Win32VS2013
	x64	Visual Studio 2013	x64Win32VS2013

2.1.2 Removed Platforms

Project files for Visual Studio 2005 are no longer provided.

2.1.3 New Command-Line Options

This release includes the following new command-line options:

❑ **-pubRate**

A new option, **-pubRate**, allows you to limit the throughput of the publisher application by specifying the number of samples per second to be sent.

❑ **-keyed**

In previous releases, to switch between using keyed and unkeyed types, you had to regenerate and recompile the test. Starting with this release, you can specify keyed types at execution time by using the new **-keyed** option. To use unkeyed types (the default), omit **-keyed**.

❑ **-executionTime**

A new option, **-executionTime**, allows you to limit the duration of the test by specifying the number of seconds to run the test.

2.1.4 Added Support for Modern API C++ Implementation

The performance test has been ported to use the new Modern C++ API implementation. In addition to the Traditional C++ API implementation, we now offer equivalent code that implements the Modern C++ API. This new executable is build as a separate project/makefile and includes the exact same functionalities as the existing language implementations.

2.1.5 Added 99.9999 Percentile in Latency Report

The publisher side of the performance test has been modified to measure up to the 99.9999 percentile for latency.

2.1.6 C++ Implementations (Traditional and Modern API) Now use High Precision Clock to Measure Time

The way in which the performance test gets timestamps has been improved for C++. Now it uses the High Precision Clock implementation provided by the *Connex DDS C* API, which will choose the most precise clock implemented in the system.

2.1.7 Removed .ini Configuration File

The performance test no longer requires a “.ini” configuration file. Now you can configure everything with command-line parameters. The .ini configuration file and its parsing have been removed. This reduces the code and now the test depends on fewer external libraries.

2.2 What's Fixed in 5.2.0

2.2.1 Degraded Performance when Using Turbo Mode

When using the command-line option **-enableTurboMode**, the resource limits **max_samples** and **max_batches** were not properly set in the *DataWriter's* QoS. This improper configuration resulted in performance degradation. This problem has been resolved.

[RTI Issue ID PERFTEST-27]

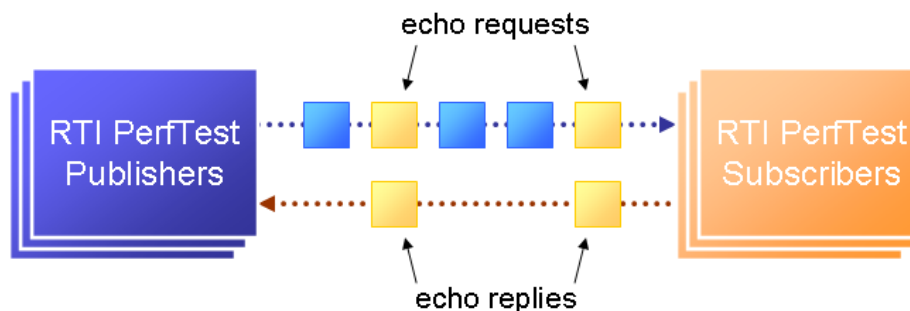
2.2.2 Potential Segmentation Fault due to Invalid Message ID

In the previous release, if an invalid value was found for **message.entity_id**, this could have caused a segmentation fault. This problem has been resolved.

[RTI Issue ID PERFTEST-76]

3 Overview

The publishing side of the test writes data as fast as it can. Every few samples (configured through the command line), it sends a special sample requesting an echo from the subscribing side. It uses this 'request -> echo' exchange to measure round-trip latency.



As you will see in [Section 8](#), there are several command-line options, including ones to designate whether the application will act as the publisher or subscriber.

You will start multiple copies of the application (typically 1 publisher and 1 or more subscribers):

- ☐ The publishing application publishes throughput data; it also subscribes to the latency echoes.
- ☐ The subscribing applications subscribe to the throughput data, in which the echo requests are embedded; they also publish the latency echoes.

The publisher prints the latency test results; the subscriber prints the throughput results.

4 Download Instructions

Download the test from the RTI Community Forum, <http://community.rti.com/>. Search for **Performance Test**.

If you need help with the download process, contact support@rti.com.

5 Installation Instructions

5.1 UNIX-Based Systems

Follow the instructions below. You do not need administrator privileges. All directory locations in this document are meant as examples only; adjust them to suit your site.

1. Make sure you have GNU's version of the tar utility, **gtar** (which handles long file names) and the unzip utility, **gunzip**. (On Linux systems, **tar** generally *is* **gtar** and **unzip** *is* **gunzip**. On Solaris systems, these utilities are separate.)
2. Use **gunzip** and **gtar** to extract the distribution into a directory of your choice. For example:


```
gunzip rtiperftestdds5xy.tar.gz
gtar xvf rtiperftestdds5xy.tar
```

The performance test files will be extracted into *<your extraction directory>/rtiperftest.5.x.y* (where 5.x.y represents the current release number).

3. Make sure the **NDDSHOME** variable is set to your *RTI Connexx DDS* installation directory, for example, **/opt/rti_connexx_dds-5.x.y**
4. Make sure the appropriate **\${NDDSHOME}/lib/<architecture>** directory is in the **LD_LIBRARY_PATH** environment variable to pick up the shared library.

The test is provided in source code only—after installation, you will need to build the application (see [Section 6](#)).

5.2 Windows Systems

Follow the instructions below. You do not need administrator privileges. All directory locations in this document are meant as examples only; adjust them to suit your site.

1. Right-click the distribution file and extract its contents into a directory of your choice.
The performance test files will be extracted into *<your extraction directory>/rtiperftest.5.x.y*.
2. Make sure the **NDDSHOME** variable is set to your *RTI Connexx DDS* installation directory, for example, **C:\Program Files\rti_connexx_dds-5.x.y**.
3. Make sure the appropriate **%NDDSHOME%\lib\<architecture>** directory is in the **Path** environment variable (so that it picks up the shared library). Or, you can copy the DLLs from **%NDDSHOME%\lib\<architecture>** into the directory where the **perftest_<language>** executable is once you've built the test (see [Building the Application \(Section 6\)](#)).

The test is provided in source code only—after installation, you will need to build the application (see [Building the Application \(Section 6\)](#)).

6 Building the Application

First, as described above, make sure the environment variable **NDDSHOME** is set to the directory containing your *RTI Connexx DDS* installation.

Examples of how to build the application:

- ❑ C++ (Modern and Traditional APIs) on a UNIX-based System

From a command-line terminal:

```
$ cd rtiperftest.5.x.y/perftest_cpp
$ gmake -f Makefile.<architecture>
$ cd rtiperftest.5.x.y/perftest_cpp2
$ gmake -f Makefile.<architecture>
```

(Where *<architecture>* is the architecture that you want to build.)

For example, to build on a Red Hat Enterprise Linux 5 32-bit system:

```
$ gmake -f Makefile.i86Linux2.6gcc4.1.1
```

Makefiles for some architectures are included in the example. If your architecture's makefile is not provided, you can create your own makefile based on the shipped example makefiles.

❑ C++ (Modern and Traditional APIs) and C# on a Windows System

1. Open the file, **perftest-*<architecture>*.sln**, in Visual Studio (where *<architecture>* is the architecture that you want to build).
2. Projects and solution files for some architectures are included in the example. If your architecture's file is not provided, you can create your own project and solution files based on the shipped example projects and solution files.
3. For better performance, select the **Mixed Platforms, Release** configuration in the Standard toolbar in Visual Studio.
4. From the Build menu, select **Build Solution**.

❑ Java on UNIX-Based or Windows Systems

You will need:

- A build tool called **Ant**, which is available free from <http://ant.apache.org/>
- JDK 1.7. The latest JDK version can be found here: <http://java.sun.com/javase/downloads/index.jsp>
- The **JAVA_HOME** environment variable must be set to the JDK installation directory (because **Ant** uses this variable).

Enter:

```
$ cd rtiperftest.5.x.y/perftest_java
$ ant -propertyfile ../resource/properties/dds_release.properties
```

7 Starting the Test

The test is provided in C++ (Modern and Traditional APIs), C#, and Java. The list below identifies how to run the executables, once you have built them, and how to pass configuration parameters to them. For detailed descriptions of the test parameters, see [Test Parameters \(Section 8\)](#). For example test configurations, see [Example Command Lines for Running the Performance Test \(Section 9\)](#).

When running the test, keep in mind that a throughput test will necessarily place a heavy load on your network and potentially on your CPU(s) as well. For the most accurate results, and the fewest complaints from your coworkers and IT department, run the test when and where you have a subnet to yourself. The test is designed to measure latency under loaded network conditions; it will produce those loads itself: there is no need to produce them externally (and your throughput results may not be meaningful if you do).

❑ The executables for C++ (Modern and Traditional APIs) and C# are in these directories:

- *<installation directory>/bin/<architecture>/Release*
- *<installation directory>/bin/<architecture>/Debug*

where *<architecture>* depends on your architecture, such as i86Linux2.6gcc4.3.4 or i86Win32VS2008.

You can differentiate the executables for the two C++ implementations (Traditional and Modern) by the name: The Traditional C++ API implementation uses **perftest_cpp** and the Modern C++ API implementation is named **perftest_cpp2**.

The test uses an XML configuration file and locates this file based on paths relative to the directory from which the test is run. Therefore, to use this configuration file:

Traditional C++:

- **bin/<architecture>/Release/perftest_cpp <-pub | -sub> [parameters] or**
- **bin/<architecture>/Debug/perftest_cpp <-pub | -sub> [parameters]**

Modern C++:

- **bin/<architecture>/Release/perftest_cpp2 <-pub | -sub> [parameters] or**
- **bin/<architecture>/Debug/perftest_cpp2 <-pub | -sub> [parameters]**

On Windows systems: As described in [Section 5.2](#), set your **Path** environment variable to the appropriate `%NDDSHOME%\lib\<architecture>` directory (this allows a shared library to be picked up when you run the executable). Or, copy the DLLs from `%NDDSHOME%\lib\<architecture>` into the directory where the **perftest_<language>** executable is located once you've built the test.

- ❑ The Java script that is used to run the Java example is in the *<installation directory>/scripts* directory.

As described in [Installation Instructions \(Section 5\)](#), you should have already set your **LD_LIBRARY_PATH** (for UNIX-based systems) or **Path** (for Windows systems) so that the application can pick up the native shared libraries on which *Connexx DDS* depends. If you have not, you can set the environment variable **RTI_PERFTEST_ARCH** to your specific architecture and let the run script set **LD_LIBRARY_PATH** or **Path** for you (assuming your **NDDSHOME** environment variable is already set).

For example:

- If you are using a Windows 32-bit architecture:
`set RTI_PERFTEST_ARCH=i86Win32jdk`
- If you are using a Windows 64-bit architecture:
`set RTI_PERFTEST_ARCH=x64Win64jdk`
- If you are using the Linux i86Linux2.6gcc4.3.4jdk architecture:
`setenv RTI_PERFTEST_ARCH i86Linux2.6gcc4.3.4jdk`

- ❑ Make sure java is in your path before running the java example run script.
- ❑ The test uses an XML configuration file. It locates this file based on its path relative to the directory from which the test is run. To use this configuration file:

- **Change to the top-level rtiperftest.<version> directory.** For example:

```
cd /opt/rti/rtiperftest.5.x.y
```

- **Start the test applications.** You can start the publisher or subscribers first, the order does not matter. When selecting your optional parameters, choose parameters that allow the test to run for *at least* 5 seconds to get any kind of meaningful results. The longer it runs, the more accurate the results will be. Ideally, you should run the test for at least 1 minute.

C++ (Traditional API):

```
bin/<architecture>/Release/perftest_cpp <-pub|-sub> [parameters]
or
bin/<architecture>/Debug/perftest_cpp <-pub|-sub> [parameters]
```

C++ (Modern API):

```
bin/<architecture>/Release/perftest_cpp2 <-pub|-sub> [parameters]
or
bin/<architecture>/Debug/perftest_cpp2 <-pub|-sub> [parameters]
```

C#:

```
bin/<architecture>/Release/perftest_cs <-pub|-sub> [parameters]
or
bin/<architecture>/Debug/perftest_cs <-pub|-sub> [parameters]
```

Java:

```
scripts/perftest_java <-pub|-sub> [parameters]
```

where *<architecture>* depends on your architecture, such as i86Linux2.6gcc4.3.4 or i86Win32VS2008.

- ❑ After the publisher recognizes that the specified number of subscribers (see the [-numSubscribers <count>](#) option) are online and the subscriber recognizes that the specified number of publishers (see the [-numPublishers <count>](#) option) are online, the test begins.

8 Test Parameters

Several parameters are available; you can enter them on the command line.

All parameters are optional and case-insensitive; partial matches are allowed (such as **-h** instead of **-help**).

Some parameters only make sense in the publishing or subscribing application. The parameters are presented in the following tables, based on whether they may be used in a publishing application, a subscribing application, or both:

- ❑ [Test Parameters for Publishing and Subscribing Applications \(Table 8.1\) on Page 9](#)
- ❑ [Test Parameters Only for Publishing Applications \(Table 8.2\) on Page 12](#)
- ❑ [Test Parameters Only for Subscribing Applications \(Table 8.3\) on Page 11](#)

As you will see in the tables, the **-pub** parameter specifies a publishing application and **-sub** specifies a subscribing application. If you do not specify **-pub** or **-sub**, **-sub** is assumed.

For additional information on setting the parameters, see:

- ❑ [Spinning vs. Sleeping \(Section 8.1\)](#)
- ❑ [Send-Queue Size and Queue-Full Behavior \(Section 8.2\)](#)
- ❑ [Number of Iterations vs. Latency Count \(Section 8.3\)](#)
- ❑ [Warming Up \(Section 8.4\)](#)
- ❑ [WaitSet Event Count and Delay \(Section 8.5\)](#)

Table 8.1 Test Parameters for Publishing and Subscribing Applications

Command-Line Option	Description
<code>-bestEffort</code>	Use best-effort communication. Default: false (use reliable communication) For an introduction to the RTI reliability model, see the Strict Reliability design pattern in the <i>RTI Connext DDS Core Libraries Getting Started Guide</i> . See also: <i>Reliable Communications</i> in the <i>RTI Connext DDS Core Libraries User's Manual</i> .
<code>-dataLen <bytes></code>	Length of payload in bytes for each send. Default: 100 bytes Range: 28 - 63,000 bytes The lower limit is the number of "overhead" bytes in the message (i.e., the timestamp, sequence number, and other meta-data used by the test); the upper limit ensures that, when the overhead of the wire protocol is added, it doesn't overflow the UDP maximum datagram size of 64KB. If <code>-scan</code> is specified, this value is ignored.
<code>-debug</code>	Run in debug mode (generates more verbose logging messages, which are useful to RTI support personnel). Default: false
<code>-durability <0 1 2 3></code>	Sets the Durability kind: <ul style="list-style-type: none"> • 0 - VOLATILE (default) • 1 - TRANSIENT LOCAL • 2 - TRANSIENT • 3 - PERSISTENT For an introduction to the RTI durability model, see the Historical Data design pattern in the <i>RTI Connext DDS Core Libraries Getting Started Guide</i> . See also: <i>Mechanisms for Achieving Information Durability and Persistence</i> , Chapter 12, in the <i>RTI Connext DDS Core Libraries User's Manual</i> .
<code>-domain <ID></code>	Domain ID. The publisher and subscriber applications must use the same domain ID in order to communicate. Default: 1 Range: 0-99 See <i>Choosing a Domain ID and Creating Multiple Domains</i> , Section 8.3.4, in the <i>RTI Connext DDS Core Libraries User's Manual</i> .
<code>-enableSharedMemory</code>	Enable the shared memory transport. Default: shared memory transport is disabled
<code>-enableTcpOnly</code>	Disable all the other transports and use only TCP transport for communication. Default: TCP transport not enabled
<code>-help</code>	Print this usage message and exit.
<code>-instanceHashBuckets <n></code>	Number of hash buckets for instances. Default: -1 (means same as the number of instances) Range: (actual value) > 0

Table 8.1 Test Parameters for Publishing *and* Subscribing Applications

Command-Line Option	Description
<code>-instances <int></code>	Set the number of instances to use in the test. The publishing and subscribing applications must specify the same number of instances. This option only makes sense when testing a keyed data type; to do so, use <code>-keyed</code> . Default: 1 Range: > 0
<code>-keepDurationUsec <usec></code>	Minimum duration that a sample is queued for ACK-disabled readers. Only used if <code>-noPositiveAcks</code> is specified on the publisher side. See <i>Disabling Positive Acknowledgements</i> , Section 6.5.3.3 in the <i>RTI Connext DDS Core Libraries User's Manual</i> . Default: 1,000 μ sec (1 millisecond) Range: ≥ 0
<code>-keyed</code>	Specify the use of a keyed type. Default: Unkeyed type.
<code>-multicast</code>	Use multicast to receive data. Default: do not use multicast.
<code>-multicastAddress <address></code>	Specify the multicast receive address for receiving user data. If unspecified, the following default values will be used according to the topic: <ul style="list-style-type: none"> latency: 239.255.1.2 throughput: 239.255.1.1 announcement: 239.255.1.100
<code>-nic <ipaddr></code>	Restrict <i>RTI Connext DDS</i> to sending output through this interface. This can be the IP address of any available network interface on the machine. By default, <i>RTI Connext DDS</i> will attempt to contact all possible subscribing nodes on all available network interfaces. Even on a multi-NIC machine, the performance over one NIC vs. another may be different (e.g., Gbit vs. 100 Mbit), so choosing the correct NIC is critical for a proper test.
<code>-noDirectCommunication</code>	Indicates if the subscribing application will receive samples from the publishing application when <i>RTI Persistence Service</i> is used. Only applies when <code>-durability <0 1 2 3></code> is TRANSIENT (2) or PERSISTENT (3). If set to true (the default), the subscribing application gets samples from the publishing application and <i>RTI Persistence Service</i> . This mode provides low latency between endpoints. If set to false, the subscribing application only gets samples from <i>RTI Persistence Service</i> . This brokered communication pattern provides a way to guarantee eventual consistency. Default: true (direct communication)
<code>-nomulticast</code>	Do <i>not</i> use multicast. Note: Starting in 5.1.0, this option is no longer needed since multicast is disabled by default. It exists only to maintain backward compatibility. Default: do not use multicast

Table 8.1 Test Parameters for Publishing *and* Subscribing Applications

Command-Line Option	Description
<code>-noPositiveAcks</code>	Disable use of positive ACKs in the reliable protocol. Default: true (use positive ACKs) See -qosprofile <filename> option for more information.
<code>-noPrintIntervals</code>	Prevent printing of statistics at intervals during the test. By default, statistics are printed every second in the subscribing application, and after receiving every latency echo in the publishing application.
<code>-qosprofile <filename></code>	Path to the XML file containing DDS QoS profiles. Default: <code>perftest.xml</code> The default file contains these QoS profiles: <ul style="list-style-type: none"> The 'ThroughputQos', 'LatencyQos', and 'AnnouncementQos' profiles are used by default. The 'NoAckThroughputQos' and 'NoAckLatencyQos' profiles are used if you specify <code>-noPositiveAcks</code>. Note: some QoS values are 'hard-coded' in the application, therefore setting them in the XML file has no effect; see the Note: on Page 15 . See comments in <code>perftest.xml</code> , as well as <i>Configuring QoS with XML</i> , Chapter 17 in the <i>RTI Connext DDS Core Libraries User's Manual</i> .
<code>-useReadThread</code>	Use a separate thread (instead of a callback) to read data. See WaitSet Event Count and Delay (Section 8.5) Default: use callback for subscriber
<code>-waitsetDelayUsec <usec></code>	Process incoming data in groups, based on time, rather than individually. Only used if the <code>-useReadThread</code> option is specified on the subscriber side. See WaitSet Event Count and Delay (Section 8.5) . Default: 100 Range: ≥ 0
<code>-waitsetEventCount <count></code>	Process incoming data in groups, based on the number of samples, rather than individually. Only used if the <code>-useReadThread</code> option is specified on the subscriber side. See WaitSet Event Count and Delay (Section 8.5) . Default: 5 Range: ≥ 1

Table 8.3 Test Parameters Only for *Subscribing* Applications

Command-Line Option	Description
<code>-numPublishers <count></code>	The subscribing application will wait for this number of publishing applications to start. Default: 1

Table 8.3 Test Parameters Only for *Subscribing* Applications

Command-Line Option	Description
<code>-sidMultiSubTest <id></code>	<p>ID of the subscriber in a multi-subscriber test.</p> <p>Use a unique value for each subscriber running on the same host that uses the same domain ID.</p> <p>Default: 0</p> <p>Range: 0 to $n-1$, inclusive, where n is the number of subscribers in a multi-subscriber test.</p>
<code>-sub</code>	<p>Set test to be a subscriber.</p> <p>Default: <code>-sub</code></p>

Table 8.2 Test Parameters Only for *Publishing* Applications

Command-Line Option	Description
<code>-batchSize <bytes></code>	<p>Enable batching and set the maximum batched message size.</p> <p>Default: 0, batching disabled</p> <p>Range: 1 to 63,000</p> <p>For more information on batching data for high throughput, see the High Throughput design pattern in the <i>RTI Connext DDS Core Libraries Getting Started Guide</i>. See also: How to Measure Latency for a Given Throughput (Section 8.6) and the BATCH QoS Policy, Section 6.5.2 in the <i>RTI Connext DDS Core Libraries Getting User's Manual</i>.</p>
<code>-enableAutoThrottle</code>	<p>Enables the Auto Throttling feature. See Auto Tuning and Turbo Mode (Section 8.7).</p> <p>Default: feature is disabled.</p>
<code>-enableTurboMode</code>	<p>Enables the Turbo Mode feature. See Auto Tuning and Turbo Mode (Section 8.7). When turbo mode is enabled, <code>-batchSize <bytes></code> is ignored.</p> <p>Default: feature is disabled.</p>
<code>-executionTime <sec></code>	<p>Allows you to limit the test duration by specifying the number of seconds to run the test.</p>
<code>-heartbeatPeriod <sec>:<nanosec></code>	<p>The period at which the publishing application will send heartbeats. See <i>Reliable Communications</i>, Chapter 10, in the <i>RTI Connext DDS Core Libraries Getting User's Manual</i>.</p> <p>Default: heartbeat period sec = 0, heartbeat period nanosec = 0 (meaning use the value as specified in the XML QoS Profile, which is set to (10 millisc = 10,000,000 nanosec)). See -qosprofile <filename>.</p> <p>Range: (actual value) 1 nanosec to 1 year (31,536,000 sec.)</p>
<code>-fastHeartbeatPeriod <sec>:<nanosec></code>	<p>An alternative heartbeat period used when the publishing application needs to flush unacknowledged samples more quickly. See <i>Reliable Communications</i>, Chapter 10, in the <i>RTI Connext DDS Core Libraries Getting User's Manual</i>.</p> <p>Default: heartbeat period sec = 0, heartbeat period nanosec = 0 (meaning use the value as specified in the XML QoS Profile, which is set to (1 millisc = 1,000,000 nanosec)). See -qosprofile <filename>.</p> <p>Range: (actual value) 1 nanosec to 1 year (31,536,000 sec). Must not be slower than <code>-heartbeatPeriod <sec>:<nanosec></code>.</p>

Table 8.2 Test Parameters Only for *Publishing* Applications

Command-Line Option	Description
<code>-latencyCount <count></code>	<p>Number samples to send before a latency ping packet is sent. See Number of Iterations vs. Latency Count (Section 8.3).</p> <p>Default: -1 (if <code>-latencyTest</code> is not specified, automatically adjust to 10,000; if <code>-latencyTest</code> is specified, automatically adjust to 1).</p> <p>Range: must be \leq <code>-numIter</code></p>
<code>-latencyTest</code>	<p>Run a latency test consisting of a ping-pong. The publisher sends a ping, then blocks until it receives a pong from the subscriber.</p> <p>Can only be used on a publisher whose <code>pidMultiPubTest = 0</code> (see <code>-pidMultiPubTest <id></code>).</p> <p>Default: false</p>
<code>-numIter <count></code>	<p>Number of samples to send. See Number of Iterations vs. Latency Count (Section 8.3) and Warming Up (Section 8.4)</p> <p>If you set <code>scan = true</code>, you cannot set this option (see <code>-scan</code>).</p> <p>Default: 0 (infinite)</p> <p>Range: <code>latencyCount</code> (adjusted value) or higher (see <code>-latencyCount <count></code>)</p>
<code>-numSubscribers <count></code>	<p>Have the publishing application wait for this number of subscribing applications to start.</p> <p>Default: 1</p>
<code>-pidMultiPubTest <id></code>	<p>Set the ID of the publisher in a multi-publisher test. Use a unique value for each publisher running on the same host that uses the same domain ID.</p> <p>Default: 0</p> <p>Range: 0 to $n-1$, inclusive, where n is the number of publishers in a multi-publisher test.</p>
<code>-pub</code>	<p>Set test to be a publisher.</p> <p>Default: <code>-sub</code></p>
<code>-pubRate</code>	<p>Limit the throughput to the specified number of samples per second.</p> <p>Default: 0 (no limit)</p> <p>Range: 1 to 10,000,000</p>
<code>-scan</code>	<p>Run test in scan mode, traversing a range of sample data sizes from 32 to 63,000 bytes.</p> <p>If you set <code>scan = true</code>, you cannot set <code>-numIter <count></code>.</p> <p>Default: false (no scan)</p>
<code>-sendQueueSize <number></code>	<p>Size of the send queue.</p> <p>When <code>-batchSize <bytes></code> is used, the size is the number of batches.</p> <p>See Send-Queue Size and Queue-Full Behavior (Section 8.2).</p> <p>Default: 50</p> <p>Range: [1,100 million] or -1 (indicating an unlimited length).</p>

Table 8.2 Test Parameters Only for *Publishing Applications*

Command-Line Option	Description
<code>-sleep <millisec></code>	Time to sleep between each send. See Spinning vs. Sleeping (Section 8.1) . Default: 0 Range: 0 or higher
<code>-spin <count></code>	Number of times to run in a spin loop between each send. See Spinning vs. Sleeping (Section 8.1) . Default: 0 Range: 0 or higher

8.1 Spinning vs. Sleeping

When the publisher is writing as fast as it can, sooner or later, it is likely to get ahead of the subscriber. There are 4 things you can do in this case:

1. Nothing—for reliable communication, `write()` will block until the subscriber(s) catch up.
2. Slow the writing down by sleeping (`-sleep <millisec>`). This approach is friendlier to the other processes on the host because it does not monopolize the CPU. However, context switching is expensive enough that you can't actually "sleep" for amounts of time on the order of microseconds, so you could end up sleeping too long and hurting performance. (Operating systems (including Linux and Windows) have a minimum resolution for sleeping; i.e., you can only sleep for a period of 1 or 10 ms. If you specify a sleep period that is less than that minimum, the OS may sleep for its minimum resolution.)
3. Spin in a tight loop between writes (`-spin <count>`). This approach will add a pause without giving up the CPU, making it easier to "sleep" for very short periods of time. In the test implementation, there is a very short loop that just performs some simple math to take up CPU time. The argument to `-spin <count>` (any number ≥ 0) is the number of times to go through that loop. The default is 0. If you specify something else, it should be a fairly large number (100's or 1000's), since spinning the loop just a few times will take negligible time. Avoid spinning on a single-core machine, as the code that would break you out of the spin may not be able to execute in a timely manner.
4. Let the publisher automatically adjust the writing rate (`-enableAutoThrottle`). This option enables the Auto Throttle feature introduced in *RTI Connext DDS 5.1.0* and its usage is preferred over `-spin <count>` because the amount of spin is automatically determined by the publisher based on the number of unacknowledged samples in the send queue.

See also: [Send-Queue Size and Queue-Full Behavior \(Section 8.2\)](#).

8.2 Send-Queue Size and Queue-Full Behavior

In many distributed systems, a data producer will often outperform data consumers. That means that, if the communications are to be reliable, the producer must be throttled in some way to allow the consumers to keep up. In some situations, this may not be a problem, because data may simply not be ready for publication at a rate sufficient to overwhelm the subscribers. If you're not so lucky, your publisher's queue of unacknowledged data will eventually fill up. When that happens, if data is not to be lost, the publication will have to block until space becomes available. Blocking can cost you in terms of latency.

To avoid the cost of blocking, consider the following:

- ❑ Enlarge your publisher's queue (`-sendQueueSize <number>`). Doing so will mean your publisher has to block less often. However, it may also let the publisher get even further ahead of slower subscribers, increasing the number of dropped and resent packets, hurting throughput. Experimenting with the send queue size is one of the easy things you can do to squeeze a little more throughput from your system.
- ❑ Enable Auto Throttling (`-enableAutoThrottle`). This option enables the Auto Throttle feature introduced in *RTI Connext DDS 5.1.0*. When this option is used, the publisher automatically adjusts the writing rate based on the number of unacknowledged samples in the send queue to avoid blocking.

Note: The following values in the `DataWriterProtocolQosPolicy` are 'hard-coded' in the application, therefore setting these values in the XML QoS profile will have no effect:

- ❑ `rtps_reliable_writer.heartbeats_per_max_samples` is set to $(\text{sendQueueSize}/10)$
- ❑ `rtps_reliable_writer.low_watermark` is set to $(\text{sendQueueSize} * 0.10)$
- ❑ `rtps_reliable_writer.high_watermark` is set to $(\text{sendQueueSize} * 0.90)$

For more information on the send queue size, see the `RESOURCE_LIMITS` QoSPolicy, Section 6.5.20 in the *RTI Connext DDS Core Libraries User's Manual* (specifically, the `max_samples` field).

8.3 Number of Iterations vs. Latency Count

When configuring the total number of samples to send during the test (`-numIter <count>`) and the number of samples to send between latency pings (`-latencyCount <count>`), keep these things in mind:

- ❑ Don't send latency pings too often. One of the purposes of the test is to measure the throughput that the middleware is able to achieve. Although the total throughput is technically the total data sent on both the throughput and latency topics, for the sake of simplicity, the test measures only the former. The implicit assumption is that the latter is negligible by comparison. If you violate this assumption, your throughput test results will not be meaningful.
- ❑ Keep the number of iterations large enough to send many latency pings over the course of the test run. (That is, keep `-numIter <count>` small compared to `-latencyCount <count>`.) Your latency measurements, and the spread between them, will be of higher quality if you are able to measure more data points.
- ❑ When selecting `-numIter <count>`, choose a value that allows the test to run for at least a minute to get accurate results. Set `-numIter <count>` to be millions for small message sizes (<1k); reduce as needed for larger sizes (otherwise the tests will take longer and longer to complete).

8.4 Warming Up

When running the performance test in Java, and to a lesser extent, C#, you may observe that throughput slowly increases through the first few incremental measurements and then levels off. This improvement reflects the background activity of the just-in-time (JIT) compiler and optimizer on these platforms. For the best indication of steady-state performance, be sure to run the test for a number of samples (`-numIter <count>`) sufficient to smooth out this start-up artifact.

8.5 WaitSet Event Count and Delay

RTI Connext DDS, and by extension, this performance test, gives you the option to either process received data in the middleware's receive thread, via a listener callback, or in a separate thread (`-useReadThread`) via an object called a WaitSet. The latter approach can be beneficial in that it

decouples the operation of your application from the middleware, so that your processing will not interfere with *Connexst DDS's* internal activities. However, it does introduce additional context switches into your data receive path. When data is arriving at a high rate, these context switches can adversely impact performance when they occur with each data sample.

To improve efficiency, the command line parameters `-waitsetDelayUsec <usec>` and `-waitsetEventCount <count>` allow you to process incoming data in groups, based on the number of samples and/or time, rather than individually, reducing the number of context switches. Experiment with these values to optimize performance for your system.

For more information, see these sections in the *RTI Connexst DDS Core Libraries User's Manual*: *Receive Threads* (Section 19.3) and *Conditions and WaitSets* (Section 4.6).

8.6 How to Measure Latency for a Given Throughput

If you want to measure the minimum latency for a given throughput, you have to use the command-line parameters `-sleep <millisec>`, `-spin <count>` and `-batchSize <bytes>` to experimentally set the throughput level for a given test run.

For example, suppose you want to generate a graph of latency vs. throughput for a packet size of 200 bytes and throughput rates of 1000, 10K, 20K, 50K, 100K, 500K, and Max messages per second.

For throughput rates under 1000 messages per second, use `-sleep <ms>` to throttle the publishing application. For example, `'-sleep 1'` will produce a throughput of approximately 1000 messages/second; `'-sleep 2'` will produce a throughput of approximately 500 messages/second.

For throughput rates higher than 1000 messages per second, use `-spin <spin count>` to cause the publishing application to busy wait between sends. The `<spin count>` value needed to produce a given throughput must be experimentally determined and is highly dependent on processor performance. For example `'-spin 19000'` may produce a message rate of 10000 messages/second with a slow processor but a rate of 14000 messages/second with a faster processor.

Use batching when you want to measure latency for throughput rates higher than the maximum rates of sending individual messages. First, determine the maximum throughput rate for the data size under test without batching (omit `-batchSize <bytes>`). For example, on a 1-Gigabyte network, for a data size of 200 bytes, the maximum throughput will be about 70,000 messages/sec. We will refer to this value as `max_no_batch`.

For all throughput rates less than `max_no_batch` (e.g., 70,000 messages/sec.), do not use batching, as this will increase the latency.

Use batching to test for throughput rates higher than `max_no_batch`: start by setting `-batchSize` to a multiple of the data size. For example, if the data size is 200 bytes, use `-batchSize 400` (this will put 2 messages in each batch), `-batchSize 800` (4 per batch), etc. This will allow you to get throughput/latency results for throughputs higher than the `max_no_batch` throughput rate, such as:

- ❑ 100,000
- ❑ 200,000
- ❑ 500,000
- ❑ millions

Note: For larger data sizes (8000 bytes and higher), batching often does not improve throughput, at least for 1-Gigabyte networks.

8.7 Auto Tuning and Turbo Mode

RTI Connex DDS 5.1.0 includes two features that allow the middleware to auto-tune the communications to achieve better performance. These features are *Auto Throttling* and *Turbo Mode*. For more information about both features, refer to Sections 10.4, *Auto Throttling for DataWriter Performance—Experimental Feature* and 6.5.2.4 *Turbo Mode: Automatically Adjusting the Number of Bytes in a Batch—Experimental Feature* in the *RTI Connex DDS Core Libraries User's Manual*. The performance test application includes two command-line options to enable these features: `-enableAutoThrottle` and `-enableTurboMode`.

With Auto Throttling, the publisher automatically adjusts the writing rate based on the number of unacknowledged samples in the send queue to avoid blocking and provide the best latency/throughput tradeoff.

With Turbo Mode, the size of a batch is automatically adjusted to provide the best latency for a given write rate. For slow write rates, the batch size will be smaller to minimize the latency penalty. For high write rates, the batch size will be bigger to increase throughput. When turbo mode is used, the command line option `-batchSize <bytes>` is ignored.

To achieve the best latency under maximum throughput conditions, use `-enableAutoThrottle` and `-enableTurboMode` in combination.

9 Example Command Lines for Running the Performance Test

The followings are examples of how to run the performance test for different use cases.

- ❑ The tests below print final results only; if you want to see intermediate values, remove the `-noPrint` argument from the command line.
- ❑ If you are running on 2 unequal machines, i.e., one machine is faster (has better processors) than another, you will see better performance by running the *Publisher* on the *slower* machine.
- ❑ To measure CPU usage while running these tests, use "top" or a similar utility.

9.1 1-1, Multicast, Best Latency as a Function of Message Size

Publisher:

```
bin/<arch>/Release/perftest_cpp -pub -noPrint -nic <ipaddr> -domain <ID>
-numIter <count> -latencyCount 1 -dataLen <length> -latencyTest
-multicast
```

Subscriber:

```
bin/<arch>/Release/perftest_cpp -sub -noPrint -nic <ipaddr> -domain <ID>
-multicast
```

- ❑ Modify `-dataLen <bytes>` to see latencies for different data sizes.
- ❑ Set `-numIter <count>` to be ≥ 100 for statistically better results.

9.2 1-1, Multicast, Maximum Throughput as a Function of Message Size (with Batching)

Publisher:

```
bin/<arch>/Release/perftest_cpp -pub -noPrint -nic <ipaddr>
-numIter <count> -dataLen <length> -batchSize <bytes>
-sendQueueSize <number> -multicast
```

Subscriber:

```
bin/<arch>/Release/perftest_cpp -sub -noPrint -nic <ipaddr> -multicast
```

- ❑ Set `-numIter <count>` to be millions for small message sizes (<1k); reduce as needed for larger sizes (otherwise the tests will take longer and longer to complete).
- ❑ To achieve maximum throughput, start by setting `-batchSize <bytes>` to 6400, then increase the size to see if you get better throughput.

The largest valid batch size is 63000 bytes.

- ❑ For maximum throughput, start by setting `-sendQueueSize <number>` to 30; the best value will usually be between 30-50.

Note: For larger data sizes (8000 bytes and higher), batching often does not improve throughput, at least for 1-Gig networks.

9.3 1-1, Multicast, Latency vs. Throughput for 200-byte Messages (with Batching)

Publisher:

```
bin/<arch>/Release/perftest_cpp -pub -noPrint -nic <ipaddr>
-numIter <count> -dataLen 200 -batchSize <bytes>
-sendQueueSize <number> -spin <count> -multicast
```

Subscriber:

```
bin/<arch>/Release/perftest_cpp -sub -noPrint -nic <ipaddr> -multicast
```

- ❑ Set `-numIter <count>` to be in the millions for high throughput tests; reduce as needed for lower throughputs (otherwise the tests will take longer and longer to complete).
- ❑ To adjust throughput, experiment with the value of `-spin <count>`. For example, to get a rate of 10,000 messages/sec, use `'-spin 20000'` to see the resulting rate, then adjust up or down as needed.

9.4 1-to-1, Multicast, Reliable UDPv4, All Sizes

Publisher:

```
bin/<arch>/Release/perftest_cpp -pub -noPrint -sendQueueSize 32
-latencyCount 10000 -scan -multicast
```

Subscriber:

```
bin/<arch>/Release/perftest_cpp -sub -noPrint -multicast
```

9.5 1-to-1, Unicast, Best-Effort, UDPv4, 1 Size

Publisher:

```
bin/<arch>/Release/perftest_cpp -pub -noPrint -sendQueueSize 32
-latencyCount 1000 -numIter 1000000 -dataLen 1024 -bestEffort
```

Subscriber:

```
bin/<arch>/Release/perftest_cpp -sub -noPrint -dataLen 1024 -bestEffort
```

9.6 1-to-1, Multicast, Reliable, UDPv4, Batching Enabled

Publisher:

```
bin/<arch>/Release/perftest_cpp -pub -noPrint -sendQueueSize 32  
-latencyCount 1000 -numIter 1000000 -dataLen 200 -batchSize 6400  
-multicast
```

Subscriber:

```
bin/<arch>/Release/perftest_cpp -sub -noPrint -dataLen 200  
-batchSize 6400 -multicast
```

9.7 1-to-2, Multicast, Reliable, UDPv4

Publisher:

```
bin/<arch>/Release/perftest_cpp -pub -noPrint -pidMultiPubTest 0  
-sendQueueSize 32 -numSubscribers 2 -latencyCount 1000  
-numIter 1000000 -dataLen 200 -multicast
```

Subscriber 1:

```
bin/<arch>/Release/perftest_cpp -sub -noPrint -dataLen 200  
-numPublishers 1 -sidMultiSubTest 0 -multicast
```

Subscriber 2:

```
bin/<arch>/Release/perftest_cpp -sub -noPrint -dataLen 200  
-numPublishers 1 -sidMultiSubTest 1 -multicast
```

9.8 2-to-1, Multicast, Reliable, UDPv4

Publisher 1:

```
bin/<arch>/Release/perftest_cpp -pub -noPrint -pidMultiPubTest 0  
-sendQueueSize 32 -numSubscribers 1 -latencyCount 1000  
-numIter 1000000 -dataLen 200 -multicast
```

Publisher 2:

```
bin/<arch>/Release/perftest_cpp -pub -noPrint -pidMultiPubTest 1  
-sendQueueSize 32 -numSubscribers 1 -latencyCount 1000  
-numIter 1000000 -dataLen 200 -multicast
```

Subscriber:

```
bin/<arch>/Release/perftest_cpp -sub -noPrint -dataLen 200  
-numPublishers 2 -sidMultiSubTest 0 -multicast
```

10 Example Output

The following is an example of the expected output from the performance test.

Publisher:

```
perftest_cpp -pub -noPrint -domain 27 -sendQueueSize 50 -latencyCount 10000 -scan
Waiting to discover 1 subscribers...
Waiting for subscribers announcement ...
Publishing data...
Length: 32 Latency: Ave 396 us Std 48.9 us Min 83 us Max 538 us 50% 401 us 90% 459 us
99% 510 us 99.99% 538 us 99.9999% 538 us
Length: 64 Latency: Ave 399 us Std 53.1 us Min 88 us Max 1062 us 50% 403 us 90% 461 us
99% 537 us 99.99% 1062 us 99.9999% 1062 us
...
```

Subscriber:

```
bin/i86Linux2.6gcc3.4.3/Release/perftest_cpp -sub -noPrint -domain 27
Waiting to discover 1 publishers ...
Waiting for data...
Length: 32 Packets: 10000000 Packets/s(ave): 47913 Mbps(ave): 12.3 Lost: 0
```

11 Optimizing Your OS for Network Performance

The network stacks of popular operating systems are not always tuned for maximum performance out of the box. RTI has found that the following configuration changes frequently improve performance for a broad set of demanding applications. Consider testing your network performance with and without these changes to learn if they can benefit your system.

11.1 Optimizing Linux Systems

Edit the file, `/etc/sysctl.conf`, and add the following:

```
net.core.wmem_max = 16777216
net.core.wmem_default = 16777216
net.core.rmem_max = 16777216
net.core.rmem_default = 16777216
net.ipv4.tcp_rmem = 4096 16777216 33554432
net.ipv4.tcp_wmem = 4096 16777216 33554432
net.ipv4.tcp_mem = 4096 16777216 33554432
run /sbin/sysctl -p
```

11.2 Optimizing Windows Systems

1. From the **Start** button, select **Run...**, then enter:
regedit
2. Change this entry: **HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters**
 - a. Add the DWORD key: **MaximumReassemblyHeaders**
 - b. Set the value to 0xffff (this is the max value)

See <http://support.microsoft.com/kb/811003> for more information.
3. Change this entry: **HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\AFD\Parameters**

- a. Add the DWORD key: **FastSendDatagramThreshold**
 - b. Set the value to 65536 (0x10000)
- See <http://support.microsoft.com/kb/235257> for more information.
4. Reboot your machine for the changes to take effect.