RTI Connext DDS

Core Libraries

Throughput Performance Test Example Using C++

Instructions

Version 5.2.3

Your systems. Working as one.

Printed in U.S.A. First printing.
April 2016.

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 Introduction 1

2 Test Operation 2

3 Paths Mentioned in Documentation 2

4 Building the Test Applications 2
4.1 Building the Test using the Makefile 3
4.2 Building the Test on a Windows Systems using a Workspace 4
4.3 Special Note for VxWorks Users using Kernel Mode 4
4.4 Special Note for VxWorks Users using RTP Mode 5

5 Command-Line Options 5

6 Example Command Line Arguments 7

7 Output Analysis 10

Throughput Test Example Using C++

This document provides instructions on using the *RTI® Connext™ DDS* throughput test example.

---

1 Introduction

In this test, the publisher sends data to one or more subscriber applications. The test goes through the following phases:

1. The publisher signals the subscriber applications that it will commence, and then starts its own clock. You may specify the duration of the test.
2. The subscriber starts measuring the number of samples received.
3. After the desired duration is over, the publisher signals the subscribers that one experiment is over. The subscriber will then divide the number of samples received by the elapsed time to report the throughput observed at the receiver. When there are multiple subscribers, the throughput reported may be different. The publisher also reports the throughput on its end.

Maximum throughput is achieved when the publisher sends as fast as the subscribers can handle messages without dropping a packet. That is, the maximum throughput is obtained somewhere between the publisher sending too slowly (not maximizing the available pipe) and the publisher swamping the subscriber (overflowing the pipe).

For this reason, the test makes the publisher try a range of sending rates; the range is provided as an input to the test. For the absolute maximum throughput to be observed, the optimal sending rate must be in the range. An ever increasing reported throughput throughout the test range is indicative that the tested range is too small. To observe the maximum throughput, the range must be extended to include faster sending rates.

By default, the message size ranges from 16 bytes to 8K where the maximum size is indicated in the IDL file.

2 Test Operation

The test code contains two applications: one for the publishing node, one for the subscribing node(s).

1. You will start the applications from the command line (publisher, then subscribers).
2. The publication application waits for the subscribing applications to announce their participation.
3. Once the publisher recognizes that the specified numbers of subscribers are online, it starts the test.
4. The test sends a burst of data, sleeps 10ms, and repeats the cycle for a specified duration. You control the amount of data and the test duration with command-line options.

3 Paths Mentioned in Documentation

The documentation refers to:

❑ **<NDDSHOME>**

This refers to the main installation directory for *Connext DDS*. The default installation paths are:

For UNIX-based systems: **/home/your user name/rti_connext_dds-*version***

For Windows systems: **C:\Program Files\rti_connext_dds-*version***

When you see **<NDDSHOME>** in the documentation, replace it with your installation path.

❑ **<path to examples>**

Examples are copied into your home directory the first time you run *RTI Launcher* or any script in **<NDDSHOME>/bin**. This document refers to the location of the copied examples as **<path to examples>.** The paths to the examples are:

For UNIX-based systems:
/home/your user name/rti_workspace/*version*/examples/

For Windows systems:
C:\Users\your user name\Documents\rti_workspace*version*\examples

When you see **<path to examples>**, replace it with the appropriate path.

4 Building the Test Applications

The example is in **<path to examples>/connext_dds/c++/performance/throughput.**

An example makefile is provided. For Windows systems, a sample Visual Studio project file is provided instead. The data type for the test is in **Throughput.idl**.

The source files are:

- **ThroughputArgs.h**, **ThroughputArgs.cxx**: Command-line argument parsing.
- **TimeManager.h**, **TimeManager.cxx**: Test duration control.
- **PerfMon.h**, **PerfMon.cxx**: On supported platforms, CPU and memory usage measurement.
- **ThroughputQos.h**, **ThroughputQos.cxx**: QoS settings for entities that are used in the test.
- **ThroughputTransport.h**, **ThroughputTransport.cxx**: Transport configuration.
- **ThroughputCommon.h**, **ThroughputCommon.cxx**: Common utilities for both the publisher and subscriber.
- **Throughput_publisher.cxx**: Publisher application.
- **Throughput_subscriber.cxx**: Subscriber application.

Regardless of the platform, **this test requires the following preprocessor defines**, in addition to the platform-specific defines described in the *RTI Connext DDS Core Libraries Platform Notes*:

- **RTI_SHARED_MEMORY**: Do not define this if your platform does not support shared memory.
- **RTI_IPV6**: Do not define this if your platform does not support IPv6 transport. (Please refer to the *RTI Connext DDS Core Libraries Release Notes* for information on using IPv6.)

The following sections provide more information about adding these defines.

4.1 Building the Test using the Makefile

1. A sample makefile is provided. To generate a makefile specific to your platform, use *rtiddsgen* with the **-example** flag, as follows:

   ```
   <NDDSHOME>/bin/rtiddsgen -example <your arch> -notypecode Throughput.idl
   ```

   Since all the source files for this example are already present in the directory, *rtiddsgen* may print messages saying that some files already exist and will not be replaced. You can safely ignore these messages.

2. Remove the newly generated **USER_QOS_PROFILES.xml** file (its default profiles are not consistent with the QoS used by the test).

3. In the generated makefile, add the preprocessor defines for shared memory and IPv6 to the **DEFINES_ARCH_SPECIFIC** variable, if your platform supports those features:

   ```
   DEFINES_ARCH_SPECIFIC = -DRTI_UNIX -DRTI_SHARED_MEMORY -DRTI_IPV6
   ```

4. Add the additional source files (**ThroughputArgs.cxx**, **TimeManager.cxx**, **PerfMon.cxx**, **ThroughputCommon.cxx**, **ThroughputQos.cxx**, **ThroughputTransport.cxx**) to the **COMMMONSOURCES** variable in the generated makefile:

   ```
   COMMONSOURCES = Throughput.cxx ThroughputPlugin.cxx \
   ThroughputSupport.cxx ThroughputArgs.cxx TimeManager.cxx \
   PerfMon.cxx ThroughputCommon.cxx ThroughputQos.cxx \
   ThroughputTransport.cxx
   ```

   See the provided makefile as an example.

5. Set the **NDDSHOME** environment variable (for more information, see the *RTI Connext DDS Core Libraries Getting Started Guide*).

6. Run this command to build:

   ```
   gmake -f makefile_Throughput_<your arch>
   ```

4.2 Building the Test on a Windows Systems using Visual Studio

1. On a Windows system, use *rtiddsgen* to generate a solution and project files for compiling the throughput example:

   ```
   <NDDSHOME>\bin\rtiddsgen -example <your arch> -notypecode Throughput.idl
   ```

   Since all the source files for this example are already present in the directory, *rtiddsgen* may print messages saying that some files already exist and will not be replaced. You can safely ignore these messages.
2. Remove the newly generated **USER_QOS_PROFILES.xml** file (its default profiles are not consistent with the QoS used by the test).
3. Add the additional source files (**ThroughputArgs.cxx**, **TimeManager.cxx**, **PerfMon.cxx**, **ThroughputCommon.cxx**, **ThroughputQos.cxx**, **ThroughputTransport.cxx**) to the publisher and subscriber projects.
   - Select the files, and drag and drop them into the **Source files** folder in the publisher and subscriber projects.
   - Alternatively in Visual Studio:
     - Right-click the **Source Files** folder under the project.
     - Select **Add**, **Add Existing Item**.
     - Select the desired files.
4. Add the preprocessor definitions **RTI_IPV6** and **RTI_SHARED_MEMORY**, if appropriate to your platform.

   In Visual Studio:
   - Right-click the project and select **Properties**.
   - In the Properties window, select the **C/C++**, **Preprocessor**.
   - Add the definitions to the Preprocessor definitions box (which should already have some defines, such as RTI_WIN32).
5. For all architectures except Windows CE: we use the PDH library to measure CPU usage, so please add **pdh.lib** as an additional link library.
6. Set the **NDDSHOME** environment variable (for more information, see the *RTI Connext DDS Core Libraries Getting Started Guide*).
7. Build your project.

4.3 Special Note for VxWorks Users using Kernel Mode

If you will be running the test on VxWorks in kernel mode, please read the following:

Since this test runs in the same process as the kernel, there is no "main" function in the publisher or subscriber. In addition, VxWorks has a limit of 10 command-line options, which precludes you from running the entry point subscriber or publisher function directly. If the preprocessor define, **RTI_VXWORKS**, is set: the publisher and subscriber are created with entry-point functions that optionally read the command-line parameters from a file. If no file is specified, the defaults will be used.

To use this functionality, create a file readable by the VxWorks OS and put your desired command line in this file. An example configuration file for a publisher would look like this:

```
-domainId 88 -nic pub_IP -transport 1 -peer sub1_IP \
-subscribers 1 -duration 10 -size 1024 -demand 1000:1000:9000
```

(Where pub_IP is the publisher's IP address and sub1_IP is the subscriber's IP address.)

Now pass the entry-point function (subscriber_main for the subscriber, or publisher_main for the publisher) the complete path to the configuration file as the only parameter.

On VxWorks 6.x systems, there is a known bug in the VxWorks kernel that manifests itself as a "memPartFree" error if the sequence of calls fopen-fread-fclose is encountered. This sequence of calls is used when reading in configuration parameters from a configuration file. If you encounter this problem, please hard-code the configuration parameters in the source files for the publisher and subscriber.

If you plan to run both the Throughput publisher and subscriber as Kernel tasks on the same VxWorks target to test loopback or shared memory, you should use dynamic linking to avoid symbol conflicts. More generally, dynamic linking should be used every time you run 2 or more DDS applications as Kernel tasks of the same VxWorks target.

4.4 Special Note for VxWorks Users using RTP Mode

If you will be running in RTP mode, the test runs in a separate process from the kernel and thus has a regular "main" function. None of the above limitations exist for RTP mode processes.

There is no high-resolution clock available for VxWorks in RTP mode, so the delay measurements cannot be as accurate as in Kernel mode. Therefore, the rate calculations provided by the Throughput test may not be as accurate as they would be in Kernel mode.

5 Command-Line Options

All options are preceded by the minus sign (**-**). Some take additional information where required. Test output can be captured using redirection on the command line.

All parameters are optional; the defaults can be found in the main function (wmain for WinCE, vx_publisher_main and vx_subscriber_main for VxWorks Kernel mode).

For examples, please see Example Command Line Arguments (Section 6).

Table 5.1 **Command-Line Options**

Option	Publishing Application	Subscribing Application
-domainId #	Sets the domain ID. Note that this test can be run at the same time as other *Connext DDS* applications, provided that the domain ID is unique.	
-participantId #	Sets the participant ID number. Use a unique value for each application running on the same host that uses the same domain ID. Publishing Application's Default: 0 Subscribing Application's Default: 1	

Table 5.1 **Command-Line Options**

Option	Publishing Application	Subscribing Application
-nic <IP>	Restricts *Connext DDS* to sending output through this interface. This can be the IP address of any available network interface on the machine. By default, *Connext DDS* will attempt to contact all possible subscribing nodes on all available network interfaces. While this may be interesting for some users, we are focusing on a simple case. 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.	
-transport #	Determines which transport to use for the test: 1 = UDP over IPv4 2 = Shared Memory 8 = UDP over IPv6	
-peer <IP address or hostname> or -peer shmem://	Specifies each peer taking part in the test. For the publishing application, specify the IP address or hostname of each subscribing application. If all subscribing applications are on the same remote host, you will only need to specify that one hostname. If each subscribing application is on a separate host, specify each one individually. If the publishing and subscribing applications are on the same host, specify that you are using shared memory: shmem:// . For the subscribing application, just specify the IP address or hostname of the publishing application. You can specify up to 16 peers; to change this limit, modify the #definition of MAX_TEST_SUBSCRIBERS in ThroughputCommon.h. When performing the test with two or more subscribing applications, the **-subscribers** flag (on the publishing side) and the **-participantId** flag (on both the publishing and subscribing sides) are both mandatory.	
-reliable	Instructs the test to use reliable communication. By default, the test uses best-effort communication.	
-mcast_recv_addr <IP>	N/A	Sets the multicast address to use for receiving data.

Table 5.1 **Command-Line Options**

Option	Publishing Application	Subscribing Application
-subscribers #	Sets the number of subscribing applications that are participating in the test. There may be more than one subscribing application (participant) on each node.	N/A
-duration #	Sets the number of seconds to be sustained for each demand run.	
-size #	Sets the payload size.	
-demand <*initial effort*>: <*incremental effort*>: <*end effort*>	Controls how many samples the publisher should write consecutively between 10ms sleep periods.	
-recoveryTime <*time in ms*>	Specifies how long to sleep after writing a specific effort (specified by `-demand`).	
-strength <*value*>	Sets the DataWriter's OwnershipStrength QoS.	
-maxBlockingTime <*time in ms*>	Sets the max_blocking_time field in the DataWriter's Reliability QoS.	
-no_push_on_write	If specified while in reliable mode, turns off push-on-write behavior. Otherwise, the writer will use push-based reliability.	
-bw_limit <*maxMbps*>	Limits the flow to the specified Mbit/s. (Only applies to large data.)	
-multicast_ttl #	Indicates the number of Time-To-Live (TTL) hops for the multicast packet. This argument needs to be used for multicast test.	

6 Example Command Line Arguments

In the following tests, the publisher is started first so that it can wait for subscribers to come online.

In Table 6.1, pub_IP, sub1_IP, sub2_IP, sub3_IP, and sub4_IP represent the IP addresses of publisher, subscriber 1, subscriber 2, subscriber 3, and subscriber 4, respectively. All IP addresses must belong to the same network. In the Scenario column, BE means Best Effort reliability, SR means Strict Reliability; 1-4 means 1 publisher to 4 subscribers.

Table 6.1 **Example Command-Line Options**

Scenario	Command-Line Options

1-1 BE 1024 bytes over shmem	**Publisher:**
	`-domainId 88 -nic pub_IP -transport 2 -peer shmem:// -subscribers 1 \`
	`-duration 10 -size 1024 -demand 1000:1000:9000`
	
	Subscriber:
	`-domainId 88 -participantId 1 -nic sub1_IP -transport 2 \ -peer shmem://`

1-4 BE 8192 bytes over shmem	**Publisher:**
	`-domainId 88 -nic pub_IP -transport 2 -peer shmem:// -subscribers 4 \`
	`-duration 10 -size 8192 -demand 1000:1000:9000`
	
	1st Subscriber:
	`-domainId 88 -participantId 1 -nic sub1_IP -transport 2 -peer shmem://`
	
	2nd Subscriber:
	`-domainId 88 -participantId 2 -nic sub2_IP -transport 2 \ -peer shmem://`
	
	3rd Subscriber:
	`-domainId 88 -participantId 3 -nic sub3_IP -transport 2 \ -peer shmem://`
	
	4th Subscriber:
	`-domainId 88 -participantId 4 -nic sub4_IP -transport 2 \ -peer shmem://`

1-1 pull SR over shmem	**Publisher:**
	`-domainId 88 -nic pub_IP -transport 2 -peer shmem:// -subscribers 1 \`
	`-duration 10 -size 1024 -demand 1000:1000:9000 -reliable \`
	`-no_push_on_write`
	
	Subscriber:
	`-domainId 88 -participantId 1 -nic sub1_IP -transport 2 \`
	`-peer shmem:// -reliable`

1-4 push SR over shmem, effort starting at 100, ending at 900, with increments of 100	**Publisher:**
	`-domainId 88 -nic pub_IP -transport 2 -peer shmem:// -subscribers 4 \`
	`-duration 10 -size 1024 -demand 100:100:900 -reliable`
	
	1st Subscriber:
	`-domainId 88 -participantId 1 -nic sub1_IP -transport 2 \`
	`-peer shmem:// -reliable`
	
	2nd Subscriber:
	`-domainId 88 -participantId 2 -nic sub2_IP -transport 2 \`
	`-peer shmem:// -reliable`
	
	3rd Subscriber:
	`-domainId 88 -participantId 3 -nic sub3_IP -transport 2 \`
	`-peer shmem:// -reliable`
	
	4th Subscriber:
	`-domainId 88 -participantId 4 -nic sub4_IP -transport 2 \`
	`-peer shmem:// -reliable`

Table 6.1 **Example Command-Line Options**

Scenario	Command-Line Options

1-1 BE over UDP unicast	**Publisher:**
	`-domainId 88 -nic pub_IP -transport 1 -peer sub1_IP -subscribers 1 \`
	`-duration 10 -size 1024 -demand 1000:1000:9000`
	
	Subscriber:
	`-domainId 88 -participantId 1 -nic sub1_IP -transport 1 -peer pub_IP`

1-4 BE over UDP unicast	**Publisher:**
	`-domainId 88 -nic pub_IP -transport 1 -peer sub1_IP -peer sub2_IP \`
	`-peer sub3_IP -peer sub4_IP -subscribers 4 -duration 10 -size 1024 \`
	`-demand 1000:1000:9000`
	
	1st Subscriber:
	`-domainId 88 -participantId 1 -nic sub1_IP -transport 1 -peer pub_IP`
	
	2nd Subscriber:
	`-domainId 88 -participantId 2 -nic sub2_IP -transport 1 -peer pub_IP`
	
	3rd Subscriber:
	`-domainId 88 -participantId 3 -nic sub3_IP -transport 1 -peer pub_IP`
	
	4th Subscriber:
	`-domainId 88 -participantId 4 -nic sub4_IP -transport 1 -peer pub_IP`

1-4 BE 32 bytes over UDP multicast	**Publisher:**
	`-domainId 88 -nic pub_IP -transport 1 -peer sub1_IP -peer sub2_IP \`
	`-peer sub3_IP -peer sub4_IP -subscribers 4 -duration 10 -size 32 \`
	`-demand 1000:1000:9000 -multicast_ttl 1`
	
	1st Subscriber:
	`-domainId 88 -participantId 1 -nic sub1_IP -transport 1 -peer pub_IP \`
	`-mcast_recv_addr 225.3.2.1`
	
	2nd Subscriber:
	`-domainId 88 -participantId 2 -nic sub2_IP -transport 1 -peer pub_IP \`
	`-mcast_recv_addr 225.3.2.1`
	
	3rd Subscriber:
	`-domainId 88 -participantId 3 -nic sub3_IP -transport 1 -peer pub_IP \`
	`-mcast_recv_addr 225.3.2.1`
	
	4th Subscriber:
	`-domainId 88 -participantId 4 -nic sub4_IP -transport 1 -peer pub_IP \`
	`-mcast_recv_addr 225.3.2.1`

7 Output Analysis

Typical output from Publisher:

```
Starting test...
bytes samples Mbit/sec duration demand CPU %  memory
----- --------- -------- -------- ------ ------ ------
 1024,   162988,  44.505,   30.00,   100,  4.67,  64.9
 1024,   208667,  56.974,   30.00,   200,  1.00,  64.9
 1024,   233846,  63.850,   30.00,   300, 10.75,  64.9
 1024,   240650,  65.710,   30.00,   400,  8.18,  64.9
 1024,   259260,  70.795,   30.00,   500, 11.08,  64.9
 1024,   263401,  71.901,   30.01,   600, 15.94,  64.9
 1024,   269965,  73.718,   30.00,   700, 13.63,  64.9
 1024,   283201,  77.330,   30.00,   800, 13.72,  64.9
 1024,   286523,  78.238,   30.00,   900, 11.87,  64.9
Test Completed.
```

Typical output from Subscriber:

```
Starting test...
bytes demand samples  lost(A) Nreject lost(N)  Mbit/s  duration CPU % memory
----- ----- -------- ------- ------- ------- -------- -------- ------ ------
 1024,  100, 162988,      0,      0,       0,  44.502,   30.00,  6.77,  62.5
 1024,  200, 208667,      0,      0,       0,  56.985,   30.00,  8.77,  62.5
 1024,  300, 233846,      0,      0,       0,  63.843,   30.01,  9.90,  62.5
 1024,  400, 240650,      0,      0,       0,  65.714,   30.00, 10.05,  62.5
 1024,  500, 259260,      0,      0,       0,  70.790,   30.00, 11.18,  62.5
 1024,  600, 263401,      0,      0,       0,  71.908,   30.01, 11.40,  62.5
 1024,  700, 269965,      0,      0,       0,  73.726,   30.00, 11.52,  62.5
 1024,  800, 283201,      0,      0,       0,  77.314,   30.01, 11.93,  62.5
 1024,  900, 286523,      0,      0,       0,  78.241,   30.00, 12.22,  62.5
Test Completed.
```

Explanation of Results:

Note that both the throughput and samples go up with demand. The CPU usage on both the publisher and the subscriber also increase.

(CPU and memory usage is measured by the PerfMon class shipped with the test. Currently, only Linux fully supports performance measurement. On Windows XP, only CPU measurement is supported. On Windows 2000, performance measurement is not supported, even though the program itself runs fine.)