RTI Connext

Core Libraries and Utilities

Latency Performance Test Example Using Java

Instructions

Version 5.1.0

Your systems. Working as one.

Printed in U.S.A. First printing.
December 2013.

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

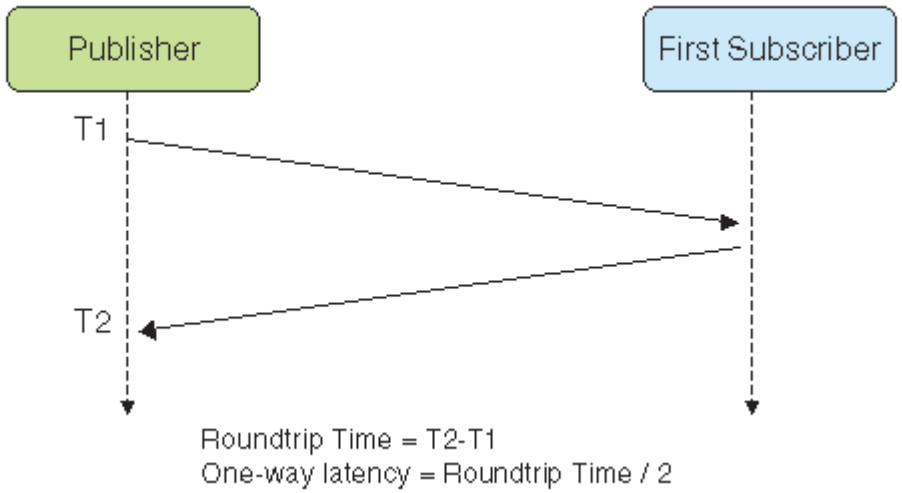
Latency Test Example Using Java

This document provides instructions on using the *RTI® Connext* (formerly *RTI Data Distribution Service*) latency test example, which is provided in source code.

1 Introduction

For a one-to-one test, one-way latency is estimated as half of the round-trip time of a message, as seen in Figure 1.

Figure 1 **One-to-One Publish-Subscribe Latency Measurement**



For a one-to-many test, one-way latency to the last subscriber is estimated by subtracting the one-way latency (as determined from a one-to-one case) from the roundtrip time to the last subscriber. Figure 2 and Figure 3 depict unicast and multicast scenarios, respectively.

Note: because of the testing methodology, for the results to be valid, the publisher and subscriber machines must be identical with respect to hardware and software setups.

Both the publisher and subscriber applications will need to be able to write and receive data. The publisher needs to be able to receive the returned data; the subscriber needs to be able to send the returned data. The publisher is the application that initiates the data stream.

Figure 2 **One-to-Two Unicast Publish -Subscribe Latency Measurement**

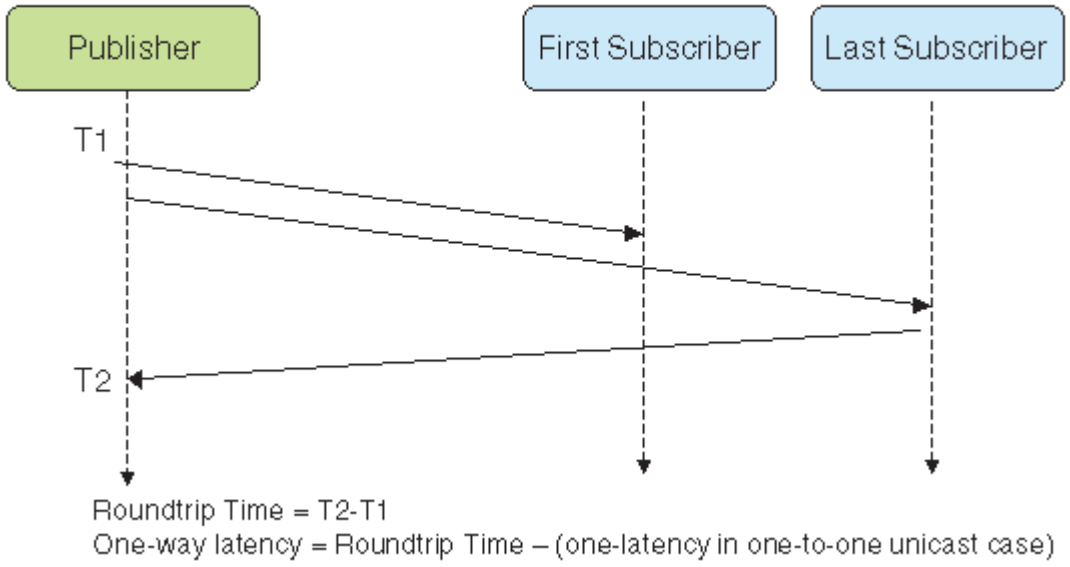
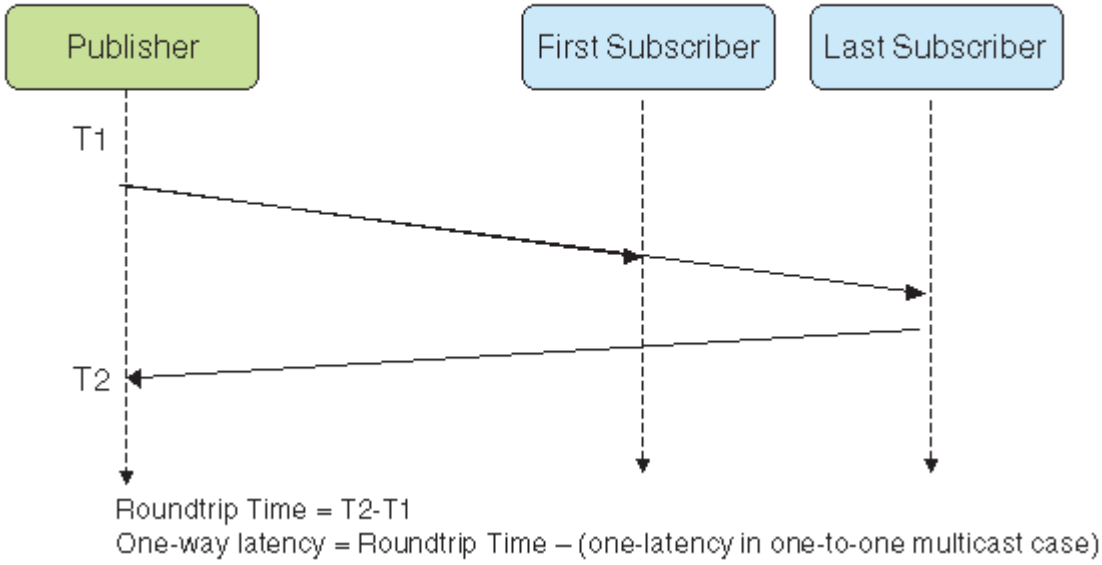


Figure 3 **One-to-Two Multicast Publish-Subscribe Latency Measurement**



By default, the message size ranges from 16 bytes to 8K, where the maximum size is indicated in the IDL file. To ensure a controlled environment for the performance test, discovery is done over unicast, not multicast.

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. For internal algorithmic reasons, the publisher must be started first, followed by each subscriber in the order of their **-cookie** (command-line option) values.
2. The publication application waits for the subscribing applications to announce their participation.
3. Once the publishing application recognizes that the specified numbers of subscribers are 'on-line,' the publisher starts publishing data.
4. After the required number of samples has been sent and the same number of replies received, the publisher signals the end of the test to all the subscribers.
5. The subscribers terminate.
6. The publisher reports the achieved latency.

3 Building the Test Applications

The example source code is found in **example/JAVA/performance/latency**.

Important: Make sure the **NDDSHOME** environment variable is set. See the *RTI Core Libraries and Utilities Getting Started Guide* for more information.

3.1 Generating a Makefile with rtiddsgen

The source code and a sample makefile are provided in the directory. To generate a makefile specific to your architecture, execute the following:

```
rtiddsgen -language Java -example <your Java arch> -notypecode Latency.idl
```

On sparcSol2.10gcc3.4.2, for example, the Java architecture is 'sparcSol2.10jdk.' *rtiddsgen* may warn you that it could not overwrite some source files because they already exist. You can safely ignore those messages.

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

3.2 Building the Test Application with the Generated Makefile

To build and run this test, you need JDK 1.5 or later.

The *rtiddsgen* utility generates **makefile_Latency_<your Java arch>**, which you can use to build the example. Please execute:

```
gmake -f makefile_Latency_<your Java arch>
```

3.3 Running the Test Using the Generated Makefile

Once the Java files are compiled, you can run the main methods of the LatencyPublisher and LatencySubscriber applications, specifying the right classpath (please see the generated makefile for the list). The generated makefile also helps you run the example by setting up the class path and providing the two targets: LatencyPublisher and LatencySubscriber.

Use the following two commands to run the publisher and subscriber applications:

```
gmake ARGS="<command-line options>" \
-f makefile_Latency_<your Java arch> LatencyPublisher

gmake ARGS="<command-line options>" \
-f makefile_Latency_<your Java arch> LatencySubscriber
```

If you have trouble running the test, please first ensure that you can run the HelloWorld example. See the *RTI Core Libraries and Utilities Getting Started Guide* and *Platform Notes* for detailed instructions.

4 Command-Line Options

Table 4.1 lists the command-line options.

All options are preceded by a minus sign (**-**). Some options 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 Options (Section 5).

5 Example Command-Line Options

On Solaris and Linux systems, set **RTI_CLASSPATH** to **$(NDDSHOME)/class/nddsjava.jar** and **LD_LIBRARY_PATH** to **$(NDDSHOME)/lib/<Java arch>**.

On Windows systems, set **RTI_CLASSPATH** to **%NDDSHOME%/class/nddsjava.jar** and **Path** to **%NDDSHOME%/lib/<Java arch>**.

Assuming you are executing the test in the **example/JAVA/performance/latency/** directory, you can start the publisher and subscribers with the following the commands:

```
gmake ARGS="<command-line options>" \
-f makefile_Latency_<your Java arch> LatencyPublisher

gmake ARGS="<command-line options>" \
-f makefile_Latency_<your Java arch> LatencySubscriber
```

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

In Table 5.1, pub_IP, sub1_IP, sub2_IP, sub3_IP, and sub4_IP represent the IP addresses of the publisher, subscriber 1, subscriber 2, subscriber 3, and subscriber 4, respectively. In the Scenario column, BE means Best Effort reliability, SR means Strict Reliability; 1-4 means 1 publisher to 4 subscribers. All IP addresses must belong to the same network.

Table 4.1 **Command-Line Options**

Option	Publishing Application	Subscribing Application
`-domainId #`	Sets the domain ID. This test can be run at the same time as other *Connext* applications, provided that the domain ID is unique.	
`-cookie #`	N/A	Sets the 'ID' of the subscriber, starting from 1. This value helps to distinguish the subscribers from each other and to the publisher. It should be unique among the subscribers and not exceed the total number of expected subscribers on the publisher side (set with `-subscribers`). The subscribers should be started in increasing order of their `-cookie` numbers. If you have two or more subscribers, this option is required, combined with `-noecho`.
`-nic <IP>`	Restricts *Connext* to sending output through this interface. This can be the IP address of any available network interface on the machine. By default, *Connext* 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.	
`-noecho`	N/A	Tells the subscriber not to echo back received data. In all test arrangements, there needs to be one and only one subscriber that will echo back data. If there are multiple subscribers, all subscribers need to specify this argument—except the subscriber with the highest `-cookie` number. If there is only one subscriber, this option should *not* be used.
`-transport #`	Determines which transport to use for the test: 1 = UDP over IPv4 2 = Shared Memory 8 = UDP over IPv6 (Please see the Known Issues section of the *RTI Core Libraries and Utilities Release Notes* for information on using IPv6.)	
`-peer <peer>`	Specifies each peer taking part in the test. For the publishing application, specify as many peers as there are subscriber hosts. For example, `-peer shmem://`, `-peer sub_IP`, `-peer sub_HostName`. For the subscribing application, you only need to specify the publisher's peer.	
`-reliable`	Tells the test to use reliable communication. By default, the test uses best-effort communication.	
`-mcast_recv_addr <IP>`	Specifies the multicast address to use for receiving data.	

Table 4.1 **Command-Line Options**

Option	Publishing Application	Subscribing Application
`-multicast_ttl #`	Indicates the number of Time-To-Live (TTL) hops for the multicast packet. This argument must be used for a multicast test.	
`-subscribers #`	Sets the number of subscribing applications participating in the test. Note that there may be more than one subscribing application (participant) on each node. This should match the largest *`-cookie`* number of the subscribers.	N/A
`-numIter #`	Sets the number of data samples to send (no limit).	
`-minSize #`	Sets the minimum size of the data packet to send. The packet size plus the additional bytes form the *Connext* packet that is sent over the network using the RTPS on-wire protocol. It is the complete packet size, excluding the protocol overhead used in the latency calculation. By default, the minimum data size is 16 bytes.	
`-maxSize #`	Sets the maximum size of the data packet to send. By default, the test will send data of up to 8K. The maximum size here must be smaller than the maximum specified by the IDL file.	

Table 5.1 **Example Command-Line Options**

Scenario	Command-Line Options

1-1 BE over shmem	**`Publisher:`**
	`-domainId 88 -nic pub_IP -transport 2 -peer shmem:// -subscribers 1 \`
	`-numIter 100000`
	
	Subscriber:
	`-domainId 88 -cookie 1 -nic sub1_IP -transport 2 -peer shmem://`

1-4 BE over shmem	**`Publisher:`**
	`-domainId 88 -nic pub_IP -transport 2 -peer shmem:// -subscribers 4 \`
	`-numIter 100000`
	
	1st Subscriber:
	`-domainId 88 -cookie 1 -nic sub1_IP -noecho -transport 2 -peer shmem://`
	
	2nd Subscriber:
	`-domainId 88 -cookie 2 -nic sub2_IP -noecho -transport 2 -peer shmem://`
	
	3rd Subscriber:
	`-domainId 88 -cookie 3 -nic sub3_IP -noecho -transport 2 -peer shmem://`
	
	4th Subscriber:
	`-domainId 88 -cookie 4 -nic sub4_IP -transport 2 -peer shmem://`

Table 5.1 **Example Command-Line Options**

Scenario	Command-Line Options

1-1 SR over shmem	**Publisher:**
	`-domainId 88 -nic pub_IP -transport 2 -peer shmem:// -subscribers 1 \ -numIter 100000 -reliable`
	
	Subscriber:
	`-domainId 88 -cookie 1 -nic sub1_IP -transport 2 -peer shmem:// -reliable`

1-4 SR over shmem	**Publisher:**
	`-domainId 88 -nic pub_IP -transport 2 \ -peer shmem:// -subscribers 4 \ -numIter 100000 -reliable`
	
	1st Subscriber:
	`-domainId 88 -cookie 1 -nic sub1_IP -noecho -transport 2 \ -peer shmem:// -reliable`
	
	2nd Subscriber:
	`-domainId 88 -cookie 2 -nic sub2_IP -noecho -transport 2 \ -peer shmem:// -reliable`
	
	3rd Subscriber:
	`-domainId 88 -cookie 3 -nic sub3_IP -noecho -transport 2 \ -peer shmem:// -reliable`
	
	4th Subscriber:
	`-domainId 88 -cookie 4 -nic sub4_IP -transport 2 -peer shmem:// -reliable`

1-1 BE over UDP unicast	**Publisher:**
	`-domainId 88 -nic pub_IP -transport 1 -peer sub1_IP \ -subscribers 1 -numIter 100000`
	
	Subscriber:
	`-domainId 88 -cookie 1 -nic sub1_IP -transport 1 -peer pub_IP`

Table 5.1 **Example Command-Line Options**

Scenario	Command-Line Options

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 -numIter 100000`
	
	1st Subscriber:
	`-domainId 88 -cookie 1 -nic sub1_IP -noecho -transport 1 -peer pub_IP`
	
	2nd Subscriber:
	`-domainId 88 -cookie 2 -nic sub2_IP -noecho -transport 1 -peer pub_IP`
	
	3rd Subscriber:
	`-domainId 88 -cookie 3 -nic sub3_IP -noecho -transport 1 -peer pub_IP`
	
	4th Subscriber:
	`-domainId 88 -cookie 4 -nic sub4_IP -transport 1 -peer pub_IP`

1-4 BE 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 -numIter 100000 \`
	`-mcast_recv_addr 225.1.2.3 -multicast_ttl 1`
	
	1st Subscriber:
	`-domainId 88 -cookie 1 -nic sub1_IP -noecho -transport 1 -peer pub_IP \`
	`-mcast_recv_addr 225.3.2.1 -multicast_ttl 1`
	
	2nd Subscriber:
	`-domainId 88 -cookie 2 -nic sub2_IP -noecho -transport 1 -peer pub_IP \`
	`-mcast_recv_addr 225.3.2.1 -multicast_ttl 1`
	
	3rd Subscriber:
	`-domainId 88 -cookie 3 -nic sub3_IP -noecho -transport 1 -peer pub_IP \`
	`-mcast_recv_addr 225.3.2.1 -multicast_ttl 1`
	
	4th Subscriber:
	`-domainId 88 -cookie 4 -nic sub4_IP -transport 1 -peer pub_IP \`
	`-mcast_recv_addr 225.3.2.1 -multicast_ttl 1`

6 Output Analysis

Typical Output from Publisher:

```
Connext Latency Test - Publisher
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Modifying RTI receive thread priorities from priority = -9999999 to -9999999
Modifying RTI database thread priorities from priority = -9999999 to -9999999
Modifying RTI event thread priorities from priority = -9999999 to -9999999
UDP enabled. Socket sendBuffer size = 8712, receiveBuffer size = 17424.
Successfully registered UDPv4 transport
Waiting for 2 receivers...[1]...[2]
Echoer restricted to the one in place #2, identified to have cookie 2.
Collecting statistics on 10000 samples per message size.
This is the roundtrip time, *not* the one-way-latency
bytes ,stdev us,ave us, min us, 50% us, 90% us, 99% us, 99.99%, max us
------,-------,-------,-------,-------,-------,-------,-------,-------
    16,   45.5,  114.0,   91.5,  104.0,  135.0,  210.0, 1790.0, 2078.5
    32,   81.1,  131.1,  101.5,  112.0,  178.0,  342.0, 1480.0, 6384.5
    64,   46.2,  118.6,  101.5,  111.0,  148.0,  206.0, 1430.0, 3006.5
   128,   46.2,  118.6,  100.5,  111.0,  134.0,  232.0, 2140.0, 2630.5
   256,   20.9,  114.4,  101.5,  111.0,  127.0,  180.0,  452.0, 1054.5
   512,   22.7,  117.4,  103.5,  113.0,  132.0,  192.0,  476.0,  511.5
  1024,   36.0,  123.5,  105.5,  114.0,  159.0,  240.0,  624.0, 1959.5
  2048,   52.8,  129.1,  108.5,  117.0,  161.0,  250.0, 2320.0, 2336.5
  4096,   32.5,  130.7,  110.5,  124.0,  132.0,  244.0, 1420.0, 1456.5
  8192,   67.1,  147.6,  126.5,  136.0,  196.0,  272.0, 2260.0, 4662.5
Test successful!
```

Typical Output from Subscriber:

```
Connext Latency Test - Subscriber
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Modifying RTI receive thread priorities from priority = -9999999 to -9999999
Modifying RTI database thread priorities from priority = -9999999 to -9999999
Modifying RTI event thread priorities from priority = -9999999 to -9999999
UDP enabled. Socket sendBuffer size = 8712, receiveBuffer size = 17424.
Successfully registered UDPv4 transport
```

Explanation of Results:

When we run a latency test, each sample varies slightly from either the average or the median. Thus, the determinism of the result can be measured by the width of the Poisson distribution. On a completely deterministic OS in an isolated network with no other network activity (which is hard to accomplish), the width should be infinitesimally small. Here are more details on each column of the results:

- ❏ Size is the size of the data packet in bytes.
- ❏ Standard deviation: On a completely deterministic system, standard deviation should be 0, and all latency values should be identical.
- ❏ Minimum latency, which shows what *Connext* is capable of doing in the best possible case. Since there is no guarantee that a run will actually hit this best case, the minimum may not increase as the payload grows. But as the sample size grows, the likelihood of such aberration decreases asymptotically.
- ❏ Latency at 50th percentile, also known as the median. Half of the samples had latency lower than this, and the other half had higher.
- ❏ Latency at 90th percentile
- ❏ Latency at 99th percentile

- Latency at 99.99th percentile, which means that only 0.01% of the samples (or 1 out of 10,000 samples) exhibited latency larger than this.
- Maximum latency