RTI Limited Bandwidth Plugins

User's Manual

Version 5.3.1

Your systems. Working as one.

Printed in U.S.A. First printing.
February 2018.

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

Chapter 1 Introduction

The *RTI* ® *Limited Bandwidth Plugins* package includes:

- Discovery Plugins
  - Limited Bandwidth Participant Discovery Plugin

    Reduces discovery time and network traffic by obtaining some of the information about the participants from an XML file instead of from the normal discovery process, which requires all information to be sent dynamically over the network. All the participants must be known ahead of time and described in an XML file.
  - Limited Bandwidth Endpoint Discovery Plugin

    Reduces discovery time and network traffic by obtaining information about the endpoints from an XML file instead of from the normal discovery process, which requires the information to be sent dynamically over the network. All the endpoints must be known ahead of time and described in an XML file.
- Transport Plugins
  - Limited Bandwidth RTPS Transport Plugin

    Reduces the size of the message headers in the Real-Time Publish Subscribe (RTPS) packages sent over the network by the *RTI Connext® DDS* software. The message headers are reduced by eliminating some fields and making other fields smaller.
  - Compression Real-Time Publish Subscribe Transport Plugin

    Compresses the RTPS packages sent over the network by *Connext DDS*. You can configure how the packages are compressed, and even provide your own compression algorithm.
- Simulation Plugin
  - Limited Bandwidth Simulation Plugin

    Simulates a Limited bandwidth shared communication channel. It includes two components:
    - A Limited Bandwidth Simulation Transport (LBST) that can be used with *Connext DDS*. This transport sends information about the RTPS packages to the LBSM (described below).
    - The Limited Bandwidth Simulation Manager (LBSM) is an external library that centralizes information about the RTPS packages that are sent; it is responsible for the simulation.

You can combine the abilities of these plugins or use them independently. When using more than one of the plugins, their properties must appear in the XML file in the order in which they should be executed.

1.1 Paths Mentioned in Documentation

The documentation refers to:

- **<NDDSHOME>**

  This refers to the installation directory for *Connext DDS*.

  The default installation paths are:

  - Mac OS X systems:

    /Applications/rti_connext_dds-*version*

  - UNIX-based systems, non-*root* user:

    /home/*your user name*/rti_connext_dds-*version*

  - UNIX-based systems, *root* user:

    /opt/rti_connext_dds-*version*

  - Windows systems, user without Administrator privileges:

    <*your home directory*>\rti_connext_dds-*version*

  - Windows systems, user with Administrator privileges:

    C:\Program Files\rti_connext_dds-*version* (for 64-bits machines) or
    C:\Program Files (x86)\rti_connext_dds-*version* (for 32-bit machines)

  You may also see $NDDSHOME or %NDDSHOME%, which refers to an environment variable set to the installation path.

  Wherever you see <NDDSHOME> used in a path, replace it with your installation path.

  Note for Windows Users: When using a command prompt to enter a command that includes the path **C:\Program Files** (or any directory name that has a space), enclose the path in quotation marks. For example:

  ```
  "C:\Program Files\rti_connext_dds-version\bin\rtiddsgen"
  ```

  or if you have defined the NDDSHOME environment variable:

  ```
  "%NDDSHOME%\bin\rtiddsgen"
  ```

- RTI Workspace directory, **rti_workspace**

  The RTI Workspace is where all configuration files for the applications and example files are located. All configuration files and examples are copied here the first time you run *RTI Launcher* or any script in **<NDDSHOME>/bin**. The default path to the RTI Workspace directory is:

  - Mac OS X systems:

    /Users/*your user name*/rti_workspace

  - UNIX-based systems:

    /home/*your user name*/rti_workspace

  - Windows systems:

    ***your Windows documents folder*\rti_workspace**

    Note: '*your Windows documents folder*' depends on your version of Windows.
    For example, on Windows 7, the folder is **C:\Users\your user name\Documents**; on Windows Server 2003, the folder is **C:\Documents and Settings\your user name\Documents**.

You can specify a different location for the **rti_workspace** directory. See the *RTI Connext DDS Core Libraries Getting Started Guide* for instructions.

- **<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 these examples as **<path to examples>.** Wherever you see <path to examples>, replace it with the appropriate path.

  By default, the examples are copied to **rti_workspace/*version*/examples**

  So the paths are:

  - Mac OS X systems:

    /Users/*your user name*/rti_workspace/*version*/examples

  - UNIX-based systems:

    /home/*your user name*/rti_workspace/*version*/examples

  - Windows systems:

    ***your Windows documents folder*\rti_workspace*version*\examples**

    Note: '*your Windows documents folder*' is described above.

  You can specify that you do not want the examples copied to the workspace. See the *RTI Connext DDS Core Libraries Getting Started Guide* for instructions.

1.2 Provided Examples

Examples are provided in these directories under **<path to examples>/connext_dds/c/limited_bandwidth_plugins**:

- dil-stacking
- lbediscovery
- lbpdiscovery
- lbrtps
- lbsm
- lbst
- zrtps

1.3 What is a Transport Plugin?

Connext DDS sends data over a variety of transport networks. *Connext DDS* has pluggable transport architecture. The core of *Connext DDS* is transport agnostic, it does not make any assumptions about the actual transports used to send and receive messages.

Connext DDS comes with standard UDPv4/IP and UDPv6/IP pluggable transports, as well as a shared memory transport; these transports are enabled by default. *Connext DDS* also give you the ability to define new transport plugins and run utilizing them.

1.4 What is Discovery?

Discovery is the behind-the-scenes way in which *Connext DDS* objects (*DomainParticipants*, *DataWriters*, and *DataReaders*) find out about each other. Each *DomainParticipant* maintains a database of information about all the active *DataReaders* and *DataWriters* in the same domain. This database is what makes it possible for *DataWriters* and *DataReaders* to communicate. To create and refresh the database, each application follows a common discovery process.

The default discovery mechanism in *Connext DDS* is the one described in the DDS specification and is known as Simple Discovery Protocol, which includes two phases: Simple Participant Discovery and Simple Endpoint Discovery. The goal of these two phases is to build, for each *DomainParticipant*, a complete picture of all the entities that belong to the remote participants in its peers list, which is a list of nodes with which a participant may communicate.

During the Simple Participant Discovery phase, *DomainParticipants* learn about each other. The *DomainParticipant's* details are communicated to all other *DomainParticipants* in the same domain by sending participant declaration messages, also known participant DATA submessages.

During the Simple Endpoint Discovery phase, *Connext DDS* matches *DataWriters* and *DataReaders*. Information about each application's *DataReaders* and *DataWriters* is exchanged by sending publication/subscription declarations in DATA submessages, which we will refer to as publication DATAs and subscription DATAs. The Simple Endpoint Discovery phase uses reliable communication.

1.5 What is Limited Bandwidth Discovery?

The Limited Bandwidth Discovery Protocol reduces the amount of information exchanged between applications. This mechanism includes two phases: Limited Bandwidth Participant Discovery and Limited Bandwidth Endpoint Discovery. Both phases can be used separately from each other. Reducing traffic on the network reduces the discovery time.

Limited Bandwidth Discovery = Limited Bandwidth Participant Discovery
+Limited Bandwidth Endpoint Discovery

1.6 Configuring Transports with the Property QoS Policy in XML

Connext DDS provides a mechanism to dynamically load an external transport from an XML QoS profile, like the file generated by *rtiddsgen* (**USER_QOS_PROFILES.xml**). The Property QoS policy is used to achieve this purpose.

The Property QoS policy stores name/value (string) pairs that can be used to configure certain parameters of *Connext DDS* that are not exposed through formal QoS policies. *Connext DDS* uses this mechanism to configure external transports.

Syntax for Setting the Property QoS Policy in an XML QoS profile:

```
<qos_library name="Property_Library">
   <qos_profile name="Property_Profile">
        <participant_qos>
             ...
             <property>
                  <value>
```

```
                        <element>
                               <name>Property1</name>
                               <value>example</value>
                        </element>
                        <element>
                               <name>Property2</name>
                               <value>example</value>
                        </element>
                        ...
                    </value>
                </property>
                ...
            </participant_qos>
        </qos_profile>
    </qos_library>
```

For more general information, see the chapter on *Configuring QoS with XML* in the *RTI Connext DDS Core Libraries User's Manual*.

For specific information on setting properties for each of the *Limited Bandwidth Plugins*, see their respective chapters in this document.

Chapter 2 Limited Bandwidth Participant Discovery Plugin

Limited Bandwidth Participant Discovery (LBPD) is achieved with a file-based plugin. Part of the information about the participants is obtained from an XML file instead of being sent dynamically over the network. This method can reduce discovery time and reduce traffic on the network. However, for LBPD to work, all the participants must be known ahead of time and described in an XML file.

The LBPD plugin reduces, but does not eliminate, the network traffic required to exchange participant information. It does this by allowing you to define some of the remote participant data, such as the product version and the RTPS protocol version, in an XML file.

The correlation between the remote participant information defined in the XML file and the information received on the network is done using the RTPS participant identifier (key) first and the participant name second if the identifier is not defined in the XML file (see Table 2.1 on page 2-2 for additional details).

This chapter describes how to configure the *RTI Limited Bandwidth Participant Discovery Plugin* and set up your *Connext DDS* application to use the plugin.

You will need two XML files, one for the discovery plugin (see Creating the LBPD Plugin Configuration File (Section 2.1)) and one for *Connext DDS* (Configuring LBPD Plugin in Connext DDS (Section 2.2)).

Note: You must link with the *dynamic* version of the *Connext DDS* libraries. See the *RTI Connext DDS Core Libraries Platform Notes* for details.

2.1 Creating the LBPD Plugin Configuration File

To use LBPD, you need an XML file that describes all the remote participants. These remote participants must be configured exactly the same as their original QoS properties.

You will specify the name of this file when you configure the plugin in the QoS Profiles XML file (**USER_QOS_PROFILES.xml**) described in Configuring LBPD Plugin in Connext DDS (Section 2.2); see “dds.discovery.participant. <*string*>.config_file” on page 2-5.

The main structure of this file is:

```
<LBPDiscoveryPluginProfile>
    <participant name="Participant1">
        ...
    </participant>
    <participant name=”Participant2">
        ...
    </participant>
</LBPDiscoveryPluginProfile>
```

Let's look at an example of this file (you can find it in **<path to examples>/connext_dds/c/limited_bandwidth_plugins/lbpdiscovery/LBPDiscoveryPluginExamplePublisher.xml**.

```
<LBPDiscoveryPluginProfile>
    <participant name="Publisher">
        <key>
            <rtps_host_id>RTPS_AUTO_ID</rtps_host_id>
            <rtps_app_id>RTPS_AUTO_ID</rtps_app_id>
            <rtps_instance_id>RTPS_AUTO_ID</rtps_instance_id>
        </key>
        <rtps_protocol_version>
            <major>2</major>
            <minor>1</minor>
        </rtps_protocol_version>
        <rtps_vendor_id>
            <vendorId>1,1</vendorId>
        </rtps_vendor_id>
        <product_version>
            <major>5</major>
            <minor>x</minor>
            <release>y</release>
            <revision>z</revision>
        </product_version>
    </participant>
</LBPDiscoveryPluginProfile>
```

(In <product_version>, *x*, *y* and *z* represent the number of the current release.)

The supported participant configuration options are described in the following tables. They are all optional.

Some descriptions also point out related *Connext DDS* documentation. For example, "See documentation on the Entity QoS policy" means you should see that section in the *RTI Connext DDS Core Libraries User's Manual* or the *Connext DDS* API Reference HTML documentation.

Table 2.1 **Configuration Properties for LBPD Plugin**

Property Name	Property Value and Description
key	The RTPS identifier of the participant. See documentation on the WireProtocol QoS policy. If a key is not set or the values are set to RTPS_AUTO_ID, the correlation between the participant information defined in the XML file and the information received from the network will be done using the participant's entity name. Example: <key> <rtps_host_id>123</rtps_host_id> <rtps_app_id>255</rtps_app_id> <rtps_instance_id>348</rtps_instance_id> </key>

Table 2.1 **Configuration Properties for LBPD Plugin**

Property Name	Property Value and Description

liveliness_lease_duration	The liveliness lease duration for the participant.
	Schema:
	`<liveliness_lease_duration>`
	`  <sec>[number | DURATION_ZERO_SEC |`
	`  DURATION_INFINITE_SEC]`
	`  </sec>`
	`  <nanosec>[number | DURATION_ZERO_NSEC |`
	`  DURATION_INFINITE_NSEC]`
	`  </nanosec>`
	`</liveliness_lease_duration>`
	Example:
	`<liveliness_lease_duration>`
	`  <sec>100</sec>`
	`  <nanosec>DURATION_ZERO_NSEC</nanosec>`
	`</liveliness_lease_duration>`

rtps_protocol_version	The version number of the RTPS protocol being used. See documentation on the ParticipantBuiltinTopicData structure.
	Example:
	`<rtps_protocol_version>`
	`  <major>2</major>`
	`  <minor>1</minor>`
	`</rtps_protocol_version>`

rtps_vendor_id	The identifier of the RTPS vendor. See documentation on ParticipantBuiltinTopicData. RTI's identifier is 1,1. Example: `<rtps_vendor_id><vendorId>1,1</vendorId><rtps_vendor_id>`

participant_name	The name of the remote participant, as set in EntityName QoS policy.
	Example: `<participant name="Participant1"/>`
	The participant name is used to correlate the information defined in the XML file with the information received on the network in the absence of the key property. If both the key property and the participant name are not defined, the discovery plugin will report the following error:
	`LBPDiscoveryPluginTypeReaderListenerOnDataAvailable:`
	`Cannot find the participant in the database`

product_version	The version number for the plugin.
	Example (where *x, y,* and *z* represent numbers of the current release):
	`<product_version>`
	`  <major>5</major>`
	`  <minor>x</minor>`
	`  <release>y</release>`
	`  <revision>z</revision>`
	`</product_version>`

2.2 Configuring LBPD Plugin in Connext DDS

This section describes how to configure the properties for the LBPD plugin in the XML QoS Profile file used by *Connext DDS* (such as **USER_QOS_PROFILES.XML**), or in the PropertyQosPolicy for your *Connext DDS* application's *DomainParticipant*.

Let's look at an example XML file, which you can find in **<path to examples>/connext_dds/c/limited_bandwidth_plugins/lbpdiscovery/USER_QOS_PROFILES.xml**:

```
<participant_qos>
  ...
  <property>
   <value>
        <!-- Specify the library -->
        <element>
             <name>dds.discovery.participant.lbpdiscovery.library</name>
             <value>rtilbpdisc</value>
        </element>

        <!-- Specify the creation function -->
        <element>
             <name>dds.discovery.participant.lbpdiscovery.create_function
             </name>
             <value>DDS_LBPDiscoveryPlugin_create</value>
        </element>

        <!-- Specify the discovery configuration file.
             Change this property to use your own file. -->
        <element>
             <name>dds.discovery.participant.lbpdiscovery.config_file</
name>
             <value>LBPDiscoveryPluginExampleSubscriber.xml</value>
        </element>

        <!-- Load LBP Participant Discovery plugin -->
        <element>
             <name>dds.discovery.participant.load_plugins</name>
             <value>dds.discovery.participant.lbpdiscovery</value>
        </element>

        <!-- Specify the verbosity -->
        <element>
             <name>dds.discovery.participant.lbpdiscovery.verbosity</name>
             <value>0</value>
        </element>
    </value>
  </property>
  ...
</participant_qos>
```

Table 2.2 describes the name/value pairs that you can use to configure the LBPD plugin.

Table 2.2 **LBPD Configuration Properties for Connext DDS**

Property Name	Property Value and Description

dds.discovery.participant.
load_plugins
	Required.
	String indicating the prefix name of the plugin that will be loaded by *Connext DDS*.
	Set the value to **dds.discovery.participant.<*string*>**, where <*string*> can be any string you want, as long as you use the same string consistently for all the properties in this table. Our example uses **lbpdiscovery:**
	`<element>`
	`<name>dds.discovery.participant.load_plugins</name>`
	`<value>dds.discovery.participant.lbpdiscovery</value>`
	`</element>`

dds.discovery.participant.
<*string*>.library
	Required.
	The name of the dynamic library that contains the LBPD plugin implementation. This library must be in the path during run time for use by *Connext DDS*.
	Set the value to **rtilbpdisc**.
	Example:
	`<element>`
	`<name>dds.discovery.participant.lbpdiscovery.library`
	`</name>`
	`<value>rtilbpdisc</value>`
	`</element>`

dds.discovery.participant.
<*string*>.create_function
	Required.
	The name of the function that will be called by *Connext DDS* to create an instance of the LBPD plugin.
	Set the value to DDS_LBPDiscoveryPlugin_create.
	Example:
	`<element>`
	`<name>`
	`dds.discovery.participant.lbpdiscovery.create_function`
	`</name>`
	`<value>DDS_LBPDiscoveryPlugin_create</value>`
	`</element>`

dds.discovery.participant.
<*string*>.config_file
	Required.
	The name of the discovery configuration file, described in Creating the LBPD Plugin Configuration File (Section 2.1).
	Set the value to the name of your own file.
	Example:
	`<element>`
	`<name>`
	`dds.discovery.participant.lbpdiscovery.config_file`
	`</name>`
	`<value>LBPDiscoveryPluginExampleSubscriber.xml</value>`
	`</element>`

Table 2.2 **LBPD Configuration Properties for Connext DDS**

Property Name	Property Value and Description

dds.discovery.participant.<*string*>.verbosity	Optional.
	The verbosity for the plugin, for debugging purposes.
	• -1: Silent
	• 0: Exceptions only (default)
	• 1: Warnings
	• 2 and up: Debug
	Example:
	`<element>`
	`  <name>`
	`  dds.discovery.participant.lbpdiscovery.verbosity`
	`  </name>`
	`  <value>0</value>`
	`</element>`

2.3 Optimizing the Plugin

You can reduce network bandwidth by changing some *Connext DDS* properties in the file **USER_QOS_PROFILE xml**. For an example of this user profile, see the file **utils/xml/USER_QOS_PROFILES.xml**.

These optimizations apply to the LBPD and LBED plugins:

- Preemptive Heartbeats and NACKs (Section 2.3.1)
- Initial Announcements (Section 2.3.2)
- Liveliness (Section 2.3.3)

2.3.1 Preemptive Heartbeats and NACKs

Each participant manages one channel with each of the other participants; the channel is used to keep the participant's liveliness. This channel is reliable and uses heartbeats and NACKs. You can reduce network bandwidth by disabling preemptive heartbeats and NACKs in this channel.

For example:

```
<participant_qos>
   <property>
        <value>
                  <element>
                       <name>
                          dds.participant.inter_participant_data_reader.
                          protocol.disable_preemptive_nack
                       </name>
                       <value>1</value>
                  </element>
                  <element>
                       <name>
                          dds.participant.inter_participant_data_writer.
                          protocol.disable_preemptive_heartbeat
                       </name>
                       <value>1</value>
```

```
                    </element>
            </value>
        </property>
    </participant_qos>
```

2.3.2 Initial Announcements

When a participant is enabled, by default it sends five announcements. You can reduce the number of initial announcements and the period between them with these properties:

- initial_participant_announcements
- min_initial_participant_announcement_period
- max_initial_participant_announcement_period

Example:

```
<participant_qos>
    <discovery_config>
        <initial_participant_announcements>
            1
        </initial_participant_announcements>
        <min_initial_participant_announcement_period>
            <sec>1</sec>
            <nanosec>0</nanosec>
        </min_initial_participant_announcement_period>
        <max_initial_participant_announcement_period>
            <sec>1</sec>
            <nanosec>0</nanosec>
        </max_initial_participant_announcement_period>
    </discovery_config>
</participant_qos>
```

2.3.3 Liveliness

The participant liveliness period can be increased with the property **participant_liveliness_assert_period**. If this property is increased, the property **participant_liveliness_lease_duration** must also be increased.

Example:

```
<participant_qos>
    <discovery_config>
        <participant_liveliness_lease_duration>
            <sec>1000</sec>
            <nanosec>DURATION_ZERO_NSEC</nanosec>
        </participant_liveliness_lease_duration>
        <participant_liveliness_assert_period>
            <sec>300</sec>
            <nanosec>DURATION_ZERO_NSEC</nanosec>
        </participant_liveliness_assert_period>
    </discovery_config>
</participant_qos>
```

Chapter 3 Limited Bandwidth Endpoint Discovery Plugin

Limited Bandwidth Endpoint Discovery (LBED) is achieved with a file-based plugin. Information about the endpoints is obtained from an XML file instead of being sent dynamically over the network. This method can reduce discovery time and reduce network traffic. However, for LBED to work, all the endpoints must be known ahead of time and described in an XML file.

This chapter describes how to configure the LBED Plugin and set up your *Connext DDS* application to use it.

You will need two XML files, one for the discovery plugin (see Creating the LBED Plugin Configuration File (Section 3.1)) and one for *Connext DDS* (Configuring LBED Plugin in Connext DDS (Section 3.2)).

Note: You must link with the *dynamic* version of the *Connext DDS* libraries. See the *RTI Connext DDS Core Libraries Platform Notes* for details.

3.1 Creating the LBED Plugin Configuration File

To use LBED, you need an XML file that describes all the remote participants and their endpoints. These remote endpoints must be configured exactly the same as their original QoS properties.

You will specify the name of this file when you configure the plugin in the QoS Profiles XML file (**USER_QOS_PROFILES.xml**) described in Configuring LBED Plugin in Connext DDS (Section 3.2); see "dds.discovery.endpoint. <*string*>.config_file" on page 3-14.

The main structure of the XML file is:

```
<LBEDiscoveryPluginProfile>
    <participant name="myParticipant">
        <datareader>
            ...
        </datareader>
        ...
        <datawriter>
            ...
        </datawriter>
    </participant>
</LBEDiscoveryPluginProfile>
```

The supported configuration options for *DataReaders* and *DataWriters* are described in Table 3.1 on page 3-3. The descriptions for many of these options also point out related *Connext DDS* documentation. For example, "See documentation on the Deadline QoS policy" means you should see that section in the *RTI Connext DDS Core Libraries User's Manual* or API Reference HTML documentation. These options can be set for *DataReaders* and *DataWriters*, unless otherwise noted in Table 3.1 on page 3-3.

The following example shows an example LBED plugin configuration file. You can find this file in **<path to examples>/connext_dds/c/limited_bandwidth_plugins/lbediscovery/LBEDiscoveryPluginExamplePublisher.xml**:

```
<?xml version="1.0"?>
<LBEDiscoveryPluginProfile xmlns:xsi=
 "http://www.w3.org/2001/XMLSchema-instance">
   <participant name="myParticipant2">
        <datawriter>
             <rtps_object_id>200</rtps_object_id>
             <topic_name>Hello LBEDiscovery</topic_name>
             <type_name>DDS::String</type_name>
             <topic_keyed>false</topic_keyed>
             <durability>
                  <kind>VOLATILE_DURABILITY_QOS</kind>
                  <direct_communication>true</direct_communication>
             </durability>

             <destination_order>
                  <kind>BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS</kind>
                  <source_timestamp_tolerance>
                       <sec>30</sec>
                       <nanosec>0</nanosec>
                  </source_timestamp_tolerance>
             </destination_order>
             <presentation>
                  <access_scope>INSTANCE_PRESENTATION_QOS</access_scope>
                  <coherent_access>false</coherent_access>
                  <ordered_access>false</ordered_access>
             </presentation>
             <deadline>
                  <period>
                       <sec>DURATION_INFINITE_SEC</sec>
                       <nanosec>DURATION_INFINITE_NSEC</nanosec>
                  </period>
             </deadline>
             <latency_budget>
                  <duration>
                       <sec>0</sec>
                       <nanosec>0</nanosec>
                  </duration>
             </latency_budget>
             <liveliness>
                  <kind>AUTOMATIC_LIVELINESS_QOS</kind>
                  <lease_duration>
                       <sec>DURATION_INFINITE_SEC</sec>
                       <nanosec>DURATION_INFINITE_NSEC</nanosec>
                  </lease_duration>
             </liveliness>
             <reliability>
                  <kind>RELIABLE_RELIABILITY_QOS</kind>
                  <max_blocking_time>
                       <sec>0</sec>
                       <nanosec>100000000</nanosec>
                  </max_blocking_time>
                  <acknowledgment_kind>
                       PROTOCOL_ACKNOWLEDGMENT_MODE
                  </acknowledgment_kind>
             </reliability>
```

```
                    <ownership>
                         <kind>EXCLUSIVE_OWNERSHIP_QOS</kind>
                    </ownership>
                    <ownership_strength>
                         <value>0</value>
                    </ownership_strength>
                    <lifespan>
                         <duration>
                              <sec>DURATION_INFINITE_SEC</sec>
                              <nanosec>DURATION_INFINITE_NSEC</nanosec>
                         </duration>
                    </lifespan>
                    <durability_service>
                         <service_cleanup_delay>
                              <sec>0</sec>
                              <nanosec>0</nanosec>
                         </service_cleanup_delay>
                         <history_kind>KEEP_LAST_HISTORY_QOS</history_kind>
                         <history_depth>1</history_depth>
                         <max_samples>LENGTH_UNLIMITED</max_samples>
                         <max_instances>LENGTH_UNLIMITED</max_instances>
                         <max_samples_per_instance>
                             LENGTH_UNLIMITED
                         </max_samples_per_instance>
                    </durability_service>
                    <service>NO_SERVICE</service>
               </datawriter>
          </participant>
     </LBEDiscoveryPluginProfile>
```

Table 3.1 **Configuration Properties for LBED Plugin**

Property Name	Property Value and Description

deadline	Maximum time between data samples. See documentation on the Deadline QoS policy.
	Schema:
	`<deadline>`
	`  <period>`
	`  <sec>[number]</sec>`
	`  <nanosec>[number]</nanosec>`
	`  </period>`
	`</deadline>`
	Example:
	`<deadline>`
	`  <period>`
	`  <sec>DURATION_INFINITE_SEC</sec>`
	`  <nanosec>DURATION_INFINITE_NSEC</nanosec>`
	`  </period>`
	`</deadline>`

Table 3.1 **Configuration Properties for LBED Plugin**

Property Name	Property Value and Description

destination_order	Controls how *Connext DDS* will deal with data sent by multiple DataWriters for the same topic. See documentation on the DestinationOrder QoS policy.
	Schema:
	`<destination_order>`
	`  <kind>[BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS |`
	`  BY_SOURCE_TIMESTAMP_DESTINATIONORDER_QOS]`
	`  </kind>`
	`  <source_timestamp_tolerance>`
	`  <sec>[number]</sec>`
	`  <nanosec>[number]</nanosec>`
	`  </source_timestamp_tolerance>`
	`</destination_order>`
	Example:
	`<destination_order>`
	`  <kind>BY_RECEPTION_TIMESTAMP_DESTINATIONORDER_QOS</kind>`
	`  <source_timestamp_tolerance>`
	`  <sec>30</sec>`
	`  <nanosec>0</nanosec>`
	`  </source_timestamp_tolerance>`
	`</destination_order>`

disable_positive_acks	Whether or not to disable positive ACKs. See documentation on the DataWriterProtocol and DataReaderProtocol QoS policies.
	Schema:
	`<protocol>`
	`  <disable_positive_acks>[true|false]</disable_positive_acks>`
	`</protocol>`
	Example:
	`<protocol>`
	`  <disable_positive_acks>true</disable_positive_acks>`
	`</protocol>`

durability	Specifies if *Connext DDS* will store and deliver previously published data to new/late-joining *DataReaders*. See documentation on the Durability QoS policy.
	Schema:
	`<durability>`
	`  <kind>`
	`  [VOLATILE_DURABILITY_QOS |`
	`  TRANSIENT_LOCAL_DURABILITY_QOS |`
	`  TRANSIENT_DURABILITY_QOS |`
	`  PERSISTENT_DURABILITY_QOS]`
	`  </kind>`
	`  <direct_communication>[true|false]</direct_communication>`
	`</durability>`
	Example:
	`<durability>`
	`  <kind>VOLATILE_DURABILITY_QOS</kind>`
	`  <direct_communication>true</direct_communication>`
	`</durability>`

Table 3.1 **Configuration Properties for LBED Plugin**

Property Name	Property Value and Description

durability_service	Various settings to configure the external *Persistence Service* used by *Connext DDS* for *DataWriters* with a Durability setting of persistent. See documentation on the Durability QoS policy.
	Schema:
	`<durability_service>`
	`  <service_cleanup_delay>`
	`  <sec>[number]</sec>`
	`  <nanosec>[number]</nanosec>`
	`  </service_cleanup_delay>`
	`  <history_kind>`
	`  [KEEP_LAST_HISTORY_QOS | KEEP_ALL_HISTORY_QOS]`
	`  </history_kind>`
	`  <history_depth>[number]</history_depth>`
	`  <max_samples>[number]</max_samples>`
	`  <max_instances>[number]</max_instances>`
	`  <max_samples_per_instance>[number]</max_samples_per_instance>`
	`</durability_service>`
	Example:
	`<durability_service>`
	`  <service_cleanup_delay>`
	`  <sec>0</sec>`
	`  <nanosec>0</nanosec>`
	`  </service_cleanup_delay>`
	`  <history_kind>KEEP_LAST_HISTORY_QOS</history_kind>`
	`  <history_depth>1</history_depth>`
	`  <max_samples>LENGTH_UNLIMITED</max_samples>`
	`  <max_instances>LENGTH_UNLIMITED</max_instances>`
	`  <max_samples_per_instance>`
	`  LENGTH_UNLIMITED`
	`  </max_samples_per_instance>`
	`</durability_service>`

group_data	Attaches arbitrary application data (a buffer of bytes) to discovery meta-data. See documentation on the GroupData QoS policy.
	Schema:
	`<group_data mode=[“hexadecimal”|”string”]>`
	`  <value>[information]</value>`
	`</group_data>`
	Example:
	`<group_data mode=”string”>`
	`  <value>Dump information</value>`
	`</group_data>`

Table 3.1 **Configuration Properties for LBED Plugin**

Property Name	Property Value and Description

latency_budget	Suggests to the middleware how much time is allowed to deliver data. See documentation on the LatencyBudget QoS policy.
	Schema:
	`<latency_budget>`
	`  <duration>`
	`  <sec>[number]</sec>`
	`  <nanosec>[number]</nanosec>`
	`  </duration>`
	`</latency_budget>`
	Example:
	`<latency_budget>`
	`  <duration>`
	`  <sec>0</sec>`
	`  <nanosec>0</nanosec>`
	`  </duration>`
	`</latency_budget>`

lifespan	Specifies how long *Connext DDS* should consider data sent by an user application to be valid. See documentation on the Lifespan QoS policy.
	Applies only to *DataWriters.*
	Schema:
	`<lifespan>`
	`  <duration>`
	`  <sec>[number]</sec>`
	`  <nanosec>[number]</nanosec>`
	`  </duration>`
	`</lifespan>`
	Example:
	`<lifespan>`
	`  <duration>`
	`  <sec>DURATION_INFINITE_SEC</sec>`
	`  <nanosec> DURATION_INFINITE_NSEC</nanosec>`
	`  </duration>`
	`</lifespan>`

Table 3.1 **Configuration Properties for LBED Plugin**

Property Name	Property Value and Description

liveliness	Liveliness specifies how *Connext DDS* determines whether a *DataWriter* is "alive." See documentation on the Liveliness QoS policy.
	Schema:
	`<liveliness>`
	`  <kind>[AUTOMATIC_LIVELINESS_QOS |`
	`  MANUAL_BY_PARTICIPANT_LIVELINESS_QOS |`
	`  MANUAL_BY_TOPIC_LIVELINESS_QOS]`
	`  </kind>`
	`  <lease_duration>`
	`  <sec>[number]</sec>`
	`  <nanosec>[number]</nanosec>`
	`  </lease_duration>`
	`</liveliness>`
	Example:
	`<liveliness>`
	`  <kind>AUTOMATIC_LIVELINESS_QOS</kind>`
	`  <lease_duration>`
	`  <sec>DURATION_INFINITE_SEC</sec>`
	`  <nanosec> DURATION_INFINITE_NSEC</nanosec>`
	`  </lease_duration>`
	`</liveliness>`

multicast	List of multicast addresses on which the *DataReader* will listen for data. See documentation on the TransportMulticast QoS policy.
	Applies only to *DataReaders.*
	Schema:
	`<multicast>`
	`  <number_of_elements>[number]</number_of_elements>`
	`  <locator>`
	`  <kind>[UDPv4]</kind>`
	`  <address>[IP address]</address>`
	`  <port>[IP port]</port>`
	`  </locator>`
	`  <locator> ...</locator>`
	`  ...`
	`</multicast>`
	Example:
	`<multicast>`
	`  <number_of_elements>2</number_of_elements>`
	`  <locator>`
	`  <kind>UDPv4</kind>`
	`  <address>192.168.1.0</address>`
	`  <port>7400</port>`
	`  </locator>`
	`  <locator>`
	`  <kind>UDPv4</kind>`
	`  <address>192.168.200.0</address>`
	`  <port>7401</port>`
	`  </locator>`
	`</multicast>`

Table 3.1 **Configuration Properties for LBED Plugin**

Property Name	Property Value and Description

ownership,
ownership_
strength
	Specifies if *DataReaders* for a topic can receive data from multiple *DataWriters* at the same time. See documentation on the Ownership and Ownership Strength QoS policies.
	Applies to *DataReaders* (except Strength option) and *DataWriters*
	Schema:
	`<ownership>`
	`<kind>[SHARED_OWNERSHIP_QOS | EXCLUSIVE_OWNERSHIP_QOS]</kind>`
	`</ownership>`
	`<ownership_strength>`
	`  <value>[number]</value>`
	`</ownership_strength>`
	Example:
	`<ownership>`
	`  <kind>EXCLUSIVE_OWNERSHIP_QOS</kind>`
	`</ownership>`
	`<ownership_strength>`
	`  <value>0</value>`
	`</ownership_strength>`

partitions	Stores a set of partition names that identify the partitions of which the entity is a member. See documentation on the Partition QoS policy.
	Schema:
	`<partitions>`
	`  <number_of_elements>[number]</number_of_elements>`
	`  <partition><name>[string]</name></partition>`
	`  <partition>...</partition>`
	`  ...`
	`</partitions>`
	Example:
	`<partitions>`
	`  <number_of_elements>2</number_of_elements>`
	`  <partition>`
	`  <name>Partition1</name>`
	`  </partition>`
	`  <partition>`
	`  <name>Partition2</name>`
	`  </partition>`
	`</partitions>`

Table 3.1 **Configuration Properties for LBED Plugin**

Property Name	Property Value and Description

presentation	Controls how *Connext DDS* presents data received by an application to the *DataReaders* of the data. See documentation on the Presentation QoS policy.
	Schema:
	`<presentation>`
	`  <access_scope>[INSTANCE_PRESENTATION_QOS |`
	`  TOPIC_PRESENTATION_QOS |`
	`  GROUP_PRESENTATION_QOS ]`
	`  </access_scope>`
	`  <coherent_access>[true|false]</coherent_access>`
	`  <ordered_access>[true|false]</ordered_access>`
	`</presentation>`
	Example:
	`<presentation>`
	`  <access_scope>INSTANCE_PRESENTATION_QOS</access_scope>`
	`  <coherent_access>false</coherent_access>`
	`  <ordered_access>false</ordered_access>`
	`</presentation>`

property	Stores name/value (string) pairs that can be used to configure certain parameters of *Connext DDS* that are not exposed through formal QoS policies. See documentation on the Property QoS policy.
	Schema:
	`<property>`
	`  <number_of_elements>[number]</number_of_elements>`
	`  <element>`
	`  <name>[string]</name>`
	`  <value>[string]</value>`
	`  <propagate>[true|false]</propagate>`
	`  </element>`
	`  <element>`
	`  ...`
	`  </element>`
	`  ...`
	`</property>`
	Example:
	`<property>`
	`  <number_of_elements>2</number_of_elements>`
	`  <element>`
	`  <name>Coin</name>`
	`  <value>Euro</value>`
	`  <propagate>false</propagate>`
	`  </element>`
	`  <element>`
	`  <name>Country</name>`
	`  <value>Spain</value>`
	`  <propagate>false</propagate>`
	`  </element>`
	`</property>`

Table 3.1 **Configuration Properties for LBED Plugin**

Property Name	Property Value and Description

reliability	Determines whether or not data published by a *DataWriter* will be reliably delivered to matching *DataReaders*. See documentation on the Reliability QoS policy.
	Schema:
	`<reliability>`
	`  <kind>[RELIABLE_RELIABILITY_QOS |`
	`  BEST_EFFORT_RELIABILITY_QOS]`
	`  </kind>`
	`  <max_blocking_time>`
	`  <sec>[number]</sec>`
	`  <nanosec>[number]</nanosec>`
	`  </max_blocking_time>`
	`  <acknowledgment_kind>[PROTOCOL_ACKNOWLEDGMENT_MODE |`
	`  APPLICATION_AUTO_ACKNOWLEDGMENT_MODE |`
	`  APPLICATION_EXPLICIT_ACKNOWLEDGMENT_MODE]`
	`  </acknowledgment_kind>`
	`</reliability>`
	Example:
	`<reliability>`
	`  <kind> RELIABLE_RELIABILITY_QOS </kind>`
	`  <max_blocking_time>`
	`  <sec>0</sec>`
	`  <nanosec>100000000</nanosec>`
	`  </max_blocking_time>`
	`  <acknowledgment_kind>`
	`  PROTOCOL_ACKNOWLEDGMENT_MODE`
	`  </acknowledgment_kind>`
	`</reliability>`

rtps_object_id	Determines the RTPS object ID of a *DataWriter*/*DataReader*. See documentation on the DataWriterProtocol QoS policy. Schema: `<rtps_object_id>[positive number]</rtps_object_id>` Example: `<rtps_object_id>1</rtps_object_id>`

rtps_protocol_ version	Version of the RTPS protocol used by the entity.
	Schema:
	`<rtps_protocol_version>`
	`  [number].[number]`
	`</rtps_protocol_version>`
	Example: `<rtps_protocol_version>0.0</rtps_protocol_version>`

service	Specifies the use of an external persistence service. Schema: `<service>[NO_SERVICE|PERSISTENCE]</service>` Example: `<service>NO_SERVICE</service>`

Table 3.1 **Configuration Properties for LBED Plugin**

Property Name	Property Value and Description

time_based_filter	Set by a *DataReader* to limit the number of new data values received over a period of time. See documentation on the TimeBasedFilter QoS policy.
	Applies only to *DataReaders.*
	Schema:
	`<time_based_filter>`
	`  <minimum_separation>`
	`  <sec>[number]</sec>`
	`  <nanosec>[number]</nanosec>`
	`  </minimum_separation>`
	`</time_based_filter>`
	Example:
	`<time_based_filter>`
	`  <minimum_separation>`
	`  <sec>0</sec>`
	`  <nanosec> 0</nanosec>`
	`  </minimum_separation>`
	`</time_based_filter>`

topic_data	Attaches arbitrary application data (a buffer of bytes) to discovery meta-data. See documentation on the TopicData QoS policy.
	Applies only to *DataReaders.*
	Schema:
	`<topic_data mode=["hexadecimal"|"string"]>`
	`  <value>[information]</value>`
	`</topic_data>`
	Example:
	`<topic_data mode="string">`
	`  <value>Dump information</value>`
	`</topic_data>`

topic_keyed	Indicates whether the data type has a key. Schema: `<topic_keyed>[true|false]</topic_keyed>` Example: `<topic_keyed>false</topic_keyed>`
topic_name	The name of the entity's topic. See documentation on the **topic** parameter in **create_datawriter()** or **create_datareader()**. Schema: `<topic_name>[string]</topic_name>` Example: `<topic_name>Example topic</topic_name>`
type_name	The name of the entity's type. See documentation on the **type_name** parameter in **create_topic()**. Schema: `<type_name>[string]</type_name>` Example: `<type_name>Example type</type_name>`

Table 3.1 **Configuration Properties for LBED Plugin**

Property Name	Property Value and Description

unicast	List of all unicast addresses on which the *DataReader/DataWriter* will listen for data. See documentation on the TransportUnicast QoS policy.
	Schema:
	`<unicast>`
	`  <number_of_elements>[number]</number_of_elements>`
	`  <locator>`
	`  <kind>[UDPv4]</kind>`
	`  <address>[IP address]</address>`
	`  <port>[IP port]</port>`
	`  </locator>`
	`  <locator> ...</locator>`
	`  ...`
	`</unicast>`
	Example:
	`<unicast>`
	`  <number_of_elements>2</number_of_elements>`
	`  <locator>`
	`  <kind>UDPv4</kind>`
	`  <address>82.123.32.4 </address><port>7400</port>`
	`  </locator>`
	`  <locator>`
	`  <kind>UDPv4</kind>`
	`  <address>82.123.32.7</address><port>7401</port>`
	`  </locator>`
	`</unicast>`

user_data	Attaches arbitrary application data (a buffer of bytes) to discovery meta-data. See documentation on the UserData QoS policy.
	Schema:
	`<user_data mode=["hexadecimal"|"string"]>`
	`  <value>[information]</value>`
	`</user_data>`
	Example:
	`<user_data mode="hexadecimal"><value>04FC8D922E</value></user_data>`

3.2 Configuring LBED Plugin in Connext DDS

This section describes how to configure the properties for the LBED plugin in the XML QoS Profile file used by *Connext DDS* (such as **USER_QOS_PROFILES.XML**), or in the PropertyQosPolicy for your application's *DomainParticipant*.

Let's look at part of an example XML file, which you can find in **<path to examples>/connext_dds/c/limited_bandwidth_plugins/lbediscovery/USER_QOS_PROFILES.xml**:

```
<qos_profile name="LBEDiscoveryPluginExampleSubscriber_Profile">
    ...
    <participant_qos>
        <property>
            <value>
                <element>
```

```
                        <name>
                            dds.discovery.endpoint.lbediscovery.library
                        </name>
                        <value>rtilbedisc</value>
                    </element>
                    <element>
                        <name>
                        dds.discovery.endpoint.lbediscovery.create_function
                        </name>
                        <value>DDS_LBEDiscoveryPlugin_create</value>
                    </element>
                    <element>
                        <name>
                            dds.discovery.endpoint.lbediscovery.config_file
                        </name>
                        <value>
                            LBEDiscoveryPluginExamplePublisher.xml
                        </value>
                    </element>
                    <element>
                        <name>dds.discovery.endpoint.load_plugins</name>
                        <value>dds.discovery.endpoint.lbediscovery</value>
                    </element>
                </value>
            </property>
        </participant_qos>
        ...
```

Table 3.2 describes the name/value pairs that you can use to configure the LBED plugin.

Table 3.2 **LBED Configuration Properties for Connext DDS**

Property Name	Property Value and Description

dds.discovery.endpoint.
load_plugins
	Required.
	String indicating the prefix name of the plugin that will be loaded by *Connext DDS*.
	Set the value to **dds.discovery.endpoint.<*string*>**, where <*string*> can be any string you want, as long as you use the same string consistently for all the properties in this table. Our example uses **lbediscovery:**
	`<element>`
	`<name>dds.discovery.endpoint.load_plugins</name>`
	`<value>dds.discovery.endpoint.lbediscovery</value>`
	`</element>`

dds.discovery.endpoint.
<*string*>.library
	Required.
	The name of the dynamic library that contains the LBED plugin implementation. This library must be in the path during run time for use by *Connext DDS*.
	Set the value to **rtilbedisc**.
	Example:
	`<element>`
	`<name>dds.discovery.endpoint.lbediscovery.library</name>`
	`<value>rtilbedisc</value>`
	`</element>`

Table 3.2 **LBED Configuration Properties for Connext DDS**

Property Name	Property Value and Description

dds.discovery.endpoint.<*string*>.create_function	Required.
	The name of the function that will be called by *Connext DDS* to create an instance of the LBED plugin.
	Set the value to DDS_LBEDiscoveryPlugin_create.
	Example:
	`<element>`
	`<name>`
	`dds.discovery.endpoint.lbediscovery.create_function`
	`</name>`
	`<value>DDS_LBEDiscoveryPlugin_create</value>`
	`</element>`

dds.discovery.endpoint.<*string*>.config_file	Required.
	The name of the discovery configuration file, described in Creating the LBED Plugin Configuration File (Section 3.1).
	Set the value to the name of your own file.
	Example:
	`<element>`
	`<name>`
	`dds.discovery.endpoint.lbediscovery.config_file`
	`</name>`
	`<value>LBEDiscoveryPluginExampleSubscriber.xml</value>`
	`</element>`

dds.discovery.endpoint.<*string*>.verbosity	The verbosity for the plugin, for debugging purposes.
	• -1: Silent
	• 0: Exceptions only (default)
	• 1: Warnings
	• 2 and up: Debug
	Example:
	`<element>`
	`<name>`
	`dds.discovery.endpoint.lbediscovery.verbosity`
	`</name>`
	`<value>0</value>`
	`</element>`

3.3 Optimizing the Plugin

You can reduce network bandwidth by changing some *Connext DDS* properties in the XML file, **USER_QOS_PROFILE xml**. An example of this user profile can be found in **<path to examples>/connext_dds/c/limited_bandwidth_plugins/lbediscovery/USER_QOS_PROFILES.xml**.

Chapter 4 Limited Bandwidth RTPS Transport Plugin

The Real-Time Publish Subscribe (RTPS) communication protocol is used by the Data Distribution Service (DDS) interoperability protocol. *Connext DDS* uses RTPS packages to send data over the network. The Limited Bandwidth RTPS (LBRTPS) transport plugin reduces the size of the message headers in the RTPS packages sent over the network. The message headers are reduced by eliminating some fields and making other fields smaller.

This chapter provides a brief overview of how RTPS messages are structured and describes how to configure the LBRTPS transport plugin. More information about RTPS can be found in the RTPS Wire Protocol Specification v2.1.

Note: You must link with the *dynamic* version of the *Connext DDS* libraries. See the *RTI Connext DDS Core Libraries Platform Notes* for details.

4.1 Understanding the RTPS Message Header

The overall structure of an RTPS *Message* consists of a fixed-size RTPS *Header* followed by a variable number of RTPS *Submessage* parts. Each *Submessage* consists of a *SubmessageHeader* and a variable number of *SubmessageElements.* The RTPS header must appear at the beginning of every message.

The *Header* contains these fields:

- **protocol:** Identifies the message as an RTPS message.

  Representation: 4 bytes

```
0.........8........16........24........32
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   'R'   |   'T'   |   'P'   |   'S'   |
+---------+---------+---------+---------+
```

- **version:** Identifies the version of the RTPS protocol. .

  Representation: 2 bytes. This release uses version {2, 1}.

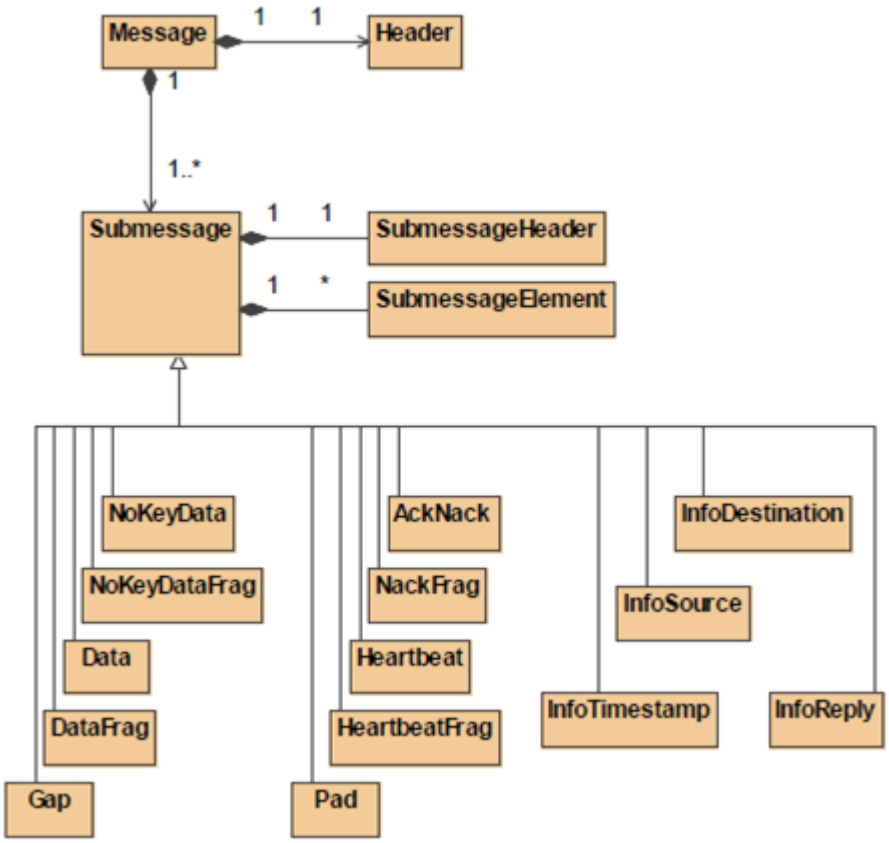
- **vendorId:** Indicates the vendor that provides the implementation of the RTPS protocol.

  Representation: 2 bytes. The RTI vendor identifier is {1, 1}.

- **guidPrefix:** Defines a default prefix to use for all GUIDs that appear in the message.

  Representation: 12 bytes: 4 bytes for the host identifier, 4 bytes for the application identifier and 4 bytes for the instance identifier.

Figure 4.1 **RTPS Message Structure**



The LBRTPS transport reduces the RTPS message headers by eliminating the **protocol**, **version**, and **vendorId** fields in the RTPS *Header* structure.

4.1.1 Submessage Structure

Each RTPS message consists of a variable number of RTPS submessages. All RTPS submessages have the same structure; they start with a *SubmessageHeader,* followed by a concatenation of *SubmessageElement* parts. The *SubmessageHeader* identifies the kind of submessage and the optional elements within that submessage.

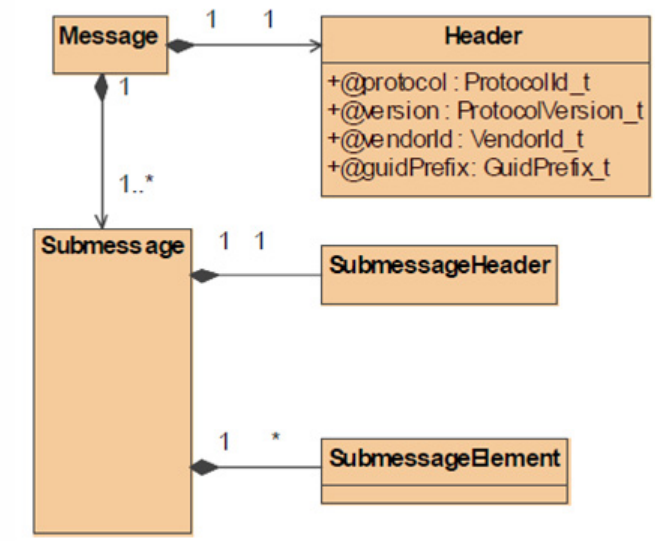
The *SubmessageHeader* contains these fields:

- **submessageId:** A 1-byte field that identifies the kind of submessage.
- **flags:** A 1-byte field that identifies the endianness used to encapsulate the submessage, the presence of optional elements within the submessage, and possibly modifies the interpretation of the submessage.
- **submessageLength:** A 2-byte field that indicates the length of the submessage. Given that an RTPS message consists of a concatenation of submessages, the submessage length can be used to skip to the next submessage.

DATA and DATA_FRAG submessages contain a field called **extraflags**. It provides space (2 bytes) for an additional 16 bits of flags beyond the 8 bits provided in the *SubmessageHeader*.

ACKNACK, HEARTBEAT, GAP, ACKNACK_FRAG, HEARTBEAT_FRAG, DATA, and DATA_FRAG submessages contain a reader entity identifier and a writer entity identifier (*readerId* field and *writerId*

Figure 4.2 **RTPS Message Header Structure**



field). The representation is 4 bytes. These submessages also contain an 8-byte field for a sequence number.

4.2 Configuring the LBRTPS Transport

The LBRTPS transport must be created using the *Connext DDS* Transport API; for more information, please see the *Transport Plugins* chapter in the *RTI Connext DDS Core Libraries User's Manual*.

Before we describe the name/value pairs that can be used in the Property QoS policy to configure the LBRTPS transport, let's review an example. The first step is to disable the builtin transports by configuring the TransportBuiltin QoS policy with the mask **MASK_NONE**. Then the name/value pairs in the Property QoS policy are set up to load and configure the LBRTPS transport plugin.

```
<qos_library name="Property_Library">
    <qos_profile name="Property_Profile">
        <participant_qos>
        ...
```

Disable built-in transports—>

```
            <transport_builtin>
                <mask>MASK_NONE</mask>
            </transport_builtin>
            <property>
                <value>
                    <element>
                      <name>dds.transport.load_plugins</name>
                      <value>dds.transport.lbrtps</value>
                    </element>
                    <element>
                      <name>dds.transport.lbrtps.library</name>
                      <value>rtilbrtps</value>
                    </element>
                    <element>
                      <name>dds.transport.lbrtps.create_function</name>
                      <value>LBRTPS_Transport_create_plugin</value>
                    </element>
```

```
                        <element>
                          <name>dds.transport.lbrtps.subtransport</name>
                          <value>UDPv4</value>
                        </element>
                        <element>
                          <name>dds.transport.lbrtps.aliases</name>
                          <value>lbrtps.udpv4</value>
                        </element>
                        <element>
                          <name>
                               dds.transport.lbrtps.UDPv4.multicast_enabled
                           </name>
                          <value>1</value>
                        </element>
                        <element>
                          <name>
                          dds.transport.lbrtps.rtps_header.eliminate_protocol
                          </name>
                          <value>true</value>
                        </element>
                        <element>
                          <name>
                           dds.transport.lbrtps.rtps_header.eliminate_version
                          </name>
                          <value>true</value>
                        </element>
                        <element>
                          <name>
                          dds.transport.lbrtps.rtps_header.eliminate_vendorId
                          </name>
                          <value>true</value>
                        </element>
                        ...
                      </value>
                    </property>
                    ...
                </participant_qos>
            </qos_profile>
        </qos_library>
```

When using the above example, *Connext DDS* will load the LBRTPS transport plugin from the library **lbrtps.dll** and call the function **LBRTPS_Transport_create_plugin()**, which will create the LBRTPS transport. The LBRTPS transport is designed to work over the *Connext DDS* UDPv4 transport. The LBRTPS transport is registered in *Connext DDS* with the participant that uses this QoS profile.

Note: *Connext DDS* does not assign initial peers to the LBRTPS transport plugin. You can set the initial peers with NDDS_DISCOVERY_PEERS, as described in the chapter on *Discovery* in the *RTI Connext DDS Core Libraries User's Manual*. The LBRTPS transport plugin example uses the NDDS_DISCOVERY_PEERS file to set the multicast address **lbrtps://239.255.0.1.**

Table 4.1 describes the name/value pairs that you can use to configure the LBRTPS transport.

4.2.1 Configuring the LBRTPS Transport Plugin's 'Subtransport' Property

The required property, **dds.transport.lbrtps.subtransport**, specifies the plugin to be loaded by the LBRTPS transport. The value can be **UDPv4**, **zrtps**, or a user-specified value, as described in the following sections. Once you set the value for a subtransport, the names of all the properties for that subtransport should be in the form **dds.transport.lbrtps.<*subtransport*>.<*property*>**.

Table 4.1 **Configuration Properties for LBRTPS Plugin**

Property Name	Property Value and Description

dds.transport.
load_plugins
	Required.
	Comma-separated strings indicating the prefix names of all plugins that will be loaded by *Connext DDS*.
	Must be set to **dds.transport.lbrtps**.
	Example:
	`<element>`
	`  <name>dds.transport.load_plugins</name>`
	`  <value>dds.transport.lbrtps</value>`
	`</element>`

dds.transport.lbrtps. subtransport	Required. Name of the plugin to be loaded by the LBRTPS transport. The LBRTPS transport will work over this loaded plugin. The value can be **UDPv4**, **zrtps**, or a user-specified string; see Configuring the LBRTPS Transport Plugin's 'Subtransport' Property (Section 4.2.1).

dds.transport.lbrtps.
library
	Required.
	The name of the dynamic library that contains the LBRTPS transport plugin implementation. This library must be in the path during run time for use by *Connext DDS*.
	Example:
	`<element>`
	`  <name>dds.transport.lbrtps.library</name>`
	`  <value>rtilbrtps</value>`
	`</element>`

dds.transport.lbrtps.
create_function
	Required.
	The name of the function that will be called by *Connext DDS* to create an instance of the LBRTPS transport. The function must have the prototype of **NDDS_Transport_create_plugin**.
	Must be set to LBRTPS_Transport_create_plugin.
	Example:
	`<element>`
	`  <name>dds.transport.lbrtps.create_function</name>`
	`  <value>LBRTPS_Transport_create_plugin</value>`
	`</element>`

dds.transport.lbrtps.
aliases
	Required.
	Aliases used to register the LBRTPS transport plugin with the *DomainParticipant*. The transport must have been created by the **dds.transport.lbrtps.create_function**. Aliases should be specified as comma-separated strings, with each comma delimiting an alias.
	An example alias for the LBRTPS transport, working over a *Connext DDS* UDPv4 transport: **lbrtps.udpv4**.
	Example:
	`<element>`
	`  <name>dds.transport.lbrtps.aliases</name>`
	`  <value>lbrtps.udpv4</value>`
	`</element>`

Table 4.1 **Configuration Properties for LBRTPS Plugin**

Property Name	Property Value and Description
The following properties are optional and appear in alphabetical order.	

dds.transport.lbrtps.
rtps_header.
eliminate_protocol
	Whether or not to eliminate the 4-byte **protocol** field in the RTPS header.
	Must be a boolean value: true or false.
	Example:
	`<element>`
	`<name>`
	`dds.transport.lbrtps.rtps_header.eliminate_protocol`
	`</name>`
	`<value>true</value>`
	`</element>`

dds.transport.lbrtps.
rtps_header.
eliminate_vendorId
	Whether or not to eliminate the 2-byte **vendorId** field in the RTPS Header.
	Must be a boolean value: true or false.
	Example:
	`<element>`
	`<name>`
	`dds.transport.lbrtps.rtps_header.eliminate_vendorId`
	`</name>`
	`<value>true</value>`
	`</element>`

dds.transport.lbrtps.
rtps_header.
eliminate_version
	Whether or not to eliminate the 2-byte **version** field in the RTPS Header.
	Must be a boolean value: true or false.
	Example:
	`<element>`
	`<name>`
	`dds.transport.lbrtps.rtps_header.eliminate_version`
	`</name>`
	`<value>true</value>`
	`</element>`

dds.transport.lbrtps. rtps_header. reduce_guidPrefix	The **reduce_guidPrefix** field is comprised of 12 bytes that represent 3 different fields of 4 bytes each: hostId, appId and instanceId. See Configuring the LBRTPS Transport Plugin's 'Subtransport' Property (Section 4.2.1)

dds.transport.lbrtps.
submessage_header.
combine_
submessageId_
with_flags
	The first two fields in the SubmessageHeader do not use all the bits. The first field, **submessageId**, only uses 5 bits. The second field, **flags**, only uses 3 bits. By setting this property to true, these two fields can be packed into a single byte.
	Must be set to a boolean value: true or false.
	Example:
	`<element>`
	`<name>`
	`dds.transport.lbrtps.submessage_header.combine_submessage-Id_with_flags`
	`</name>`
	`<value>true </value>`
	`</element>`

Table 4.1 **Configuration Properties for LBRTPS Plugin**

Property Name	Property Value and Description

dds.transport.lbrtps.
submessages.
reduce_entitiesId
	Many submessage kinds include two fields: **readerId** and **writerId**.
	In the RTPS protocol specification, these fields are mapped to structure:
	`struct {`
	`  octet[3] entityKey;`
	`  octet entityKind;`
	`} EntityId_t;`
	As you can see, 4 bytes are allocated for each entity: 3 for the entityKey and 1 for the entityKind. However, you may be able to reduce the size of these fields if you know ahead of time how many DataReaders and DataWriters you will have.
	If you will have no more than 2048 DataReaders and 2048 DataWriters, you can reduce the size of each of these fields from four to two bytes. And if you will have no more than 8 DataReaders and 8 DataWriters, you can reduce each field to only one byte. The math involved is explained below.
	Five bits are always needed for the entityKind. If you have no more than 2048 DataReaders and 2048 DataWriters, their entityKeys can be 0-2047, which will fit in 11 bits. Thus you only need 2 bytes in this case: 5 bits for entityKind + 11 bits for entityKey = 16 bits = 2 bytes.
	Suppose you have no more than 8 DataReaders and 8 DataWriters. In this case, you still need 5 bits for the entityKind, but only 3 bits to hold entityKeys 0-7. So you would only need 8 bits: 5 bits for entityKind + 3 bits for entityKey = 8 bits = 1 byte.
	This property's value is expressed as 2 comma-separated integers between 8 and 32, to specify the number of bits to use for the readerId and writerId.
	For example, to reduce both readerId and writerId to 12 bits each:
	`<element>`
	`  <name>dds.transport.lbrtps.submessages.reduce_entitiesId`
	`  </name>`
	`  <value>12,12</value>`
	`</element>`

dds.transport.lbrtps.
submessages.
reduce_
sequenceNumber
	Some submessage kinds keep track of the number of submessages received by using an 8-byte sequence number field. You can reduce the number of bytes used for the sequence number by using this property.
	The value must be an integer between 16 and 64 that specifies the desired size of the sequence number, in bits.
	Example:
	`<element>`
	`  <name>`
	`  dds.transport.lbrtps.submessages.reduce_sequenceNumber`
	`  </name>`
	`  <value>32</value>`
	`</element>`

dds.transport.lbrtps.
verbosity
	The verbosity for the plugin, for debugging purposes.
	• -1: Silent
	• 0: Exceptions only (default)
	• 1: Warnings
	• 2 and up: Debug
	Example:
	`<element>`
	`  <name>dds.transport.lbrtps.verbosity>/name>`
	`  <value>0</value>`
	`</element>`

4.2.1.1 Using UDPv4 as a Subtransport

To load the *Connext DDS* UDPv4 built-in transport, use the value **UDPv4**. If you want the UDPv4 transport to be created with multicast support, also set **dds.transport.lbrtps.UDPv4.multicast_enabled** to 1, as seen in the example below.

To use the UDPv4 transport and enable multicast:

```
<element>
    <name>dds.transport.lbrtps.subtransport</name>
    <value>UDPv4</value>
</element>
<element>
    <name>dds.transport.lbrtps.UDPv4.multicast_enabled</name>
    <value>1</value>
</element>
```

4.2.1.2 Using ZRTPS as a Subtransport

In Chapter 5, you will learn about the Compression Real-Time Publish Subscribe Transport Plugin (ZRTPS). You can use the LBRTPS and ZRTPS transport plugins together, as long as you meet the following three requirements. By using this combination of transport plugins, the LBRTPS transport plugin will reduce the RTPS headers, then the ZRTPS transport plugin will compress the RTPS package.

1. The LBRTPS transport plugin properties must appear in the XML QoS profile *before* those for the ZRTPS transport plugin. (See the example below.)

   As mentioned in Chapter 1, the plugins are executed in the order in which they appear in the XML file. So having the LBRTPS properties appear in the file before the ZRTPS properties is important because once the message is compressed by the ZRTPS transport plugin, the header is no longer recognizable by the LBRTPS transport plugin as an RTPS header.

2. The ZRTPS transport plugin must be configured as a subtransport of the LBRTPS transport plugin using the value **zrtps**, as seen here:

   ```
   <element>
           <name>dds.transport.lbrtps.subtransport</name>
           <value>zrtps</value>
   </element>
   ```

   You will also need to set two additional properties:

   - dds.transport.lbrtps.zrtps.library
   - dds.transport.lbrtps.zrtps.create_function

   ```
   <element>
       <name>dds.transport.lbrtps.zrtps.library</name>
       <value>rtizrtps</value>
   </element>

   <element>
       <name>dds.transport.lbrtps.zrtps.create_function</name>
       <value>ZRTPS_Transport_create_plugin</value>
   </element>
   ```

3. The LBRTPS and ZRTPS transport plugins cannot both use UDPv4 as a subtransport because of port conflict issues.

The following example configures the LBRTPS transport plugin with ZRTPS as a subtransport.

```
<!-- LBRTPS -->
<element>
    <name>dds.transport.load_plugins</name>
```

```
    <value>dds.transport.lbrtps</value>
</element>
<element>
    <name>dds.transport.lbrtps.library</name>
    <value>rtilbrtps</value>
</element>
<element>
    <name>dds.transport.lbrtps.create_function</name>
    <value>LBRTPS_Transport_create_plugin</value>
</element>
<element>
    <name>dds.transport.lbrtps.aliases</name>
    <value>lbrtps.zrtps</value>
</element>
<element>
    <name>dds.transport.lbrtps.subtransport</name>
    <value>zrtps</value>
</element>
<element>
    <name>dds.transport.lbrtps.verbosity</name>
    <value>2</value>
</element>

<!-- ZRTPS-->
<element>
    <name>dds.transport.lbrtps.zrtps.library</name>
    <value>rtizrtps</value>
</element>
<element>
    <name>dds.transport.lbrtps.zrtps.create_function</name>
    <value>ZRTPS_Transport_create_plugin</value>
</element>
<element>
    <name>dds.transport.lbrtps.zrtps.subtransport</name>
    <value>UDPv4</value>
</element>
<element>
    <name>dds.transport.lbrtps.zrtps.UDPv4.multicast_enabled</name>
    <value>1</value>
</element>
<element>
    <name>dds.transport.lbrtps.zrtps.compression_library</name>
    <value>AUTOMATIC_COMPRESSION</value>
</element>
<element>
    <name>dds.transport.lbrtps.zrtps.compression_level</name>
    <value>9</value>
</element>
<element>
    <name>dds.transport.lbrtps.zrtps.verbosity</name>
    <value>0</value>
</element>
```

4.2.1.3 Using a User-Specified Subtransport

To load a user-provided transport plugin, provide a value for **dds.transport.lbrtps.subtransport,** and then use that same value like a prefix to define these two additional properties:

- dds.transport.lbrtps.*prefix*.library

- dds.transport.lbrtps.*prefix*.create_function

For example, to specify the value, **testplugin**:

```
<element>
    <name>dds.transport.lbrtps.subtransport</name>
    <value>testplugin</value>
</element>
<element>
    <name>dds.transport.lbrtps.testplugin.library</name>
    <value>testplugin</value>
</element>
<element>
    <name>dds.transport.lbrtps.testplugin.create_function</name>
    <value>TestPlugin_Transport_create_function</value>
</element>
```

4.2.2 Configuring the LBRTPS Transport Plugin's 'reduce_guidPrefix' Property

The **dds.transport.lbrtps.rtps_header.reduce_guidPrefix** property is comprised of 12 bytes that represent three different 4-byte fields:

- **hostId**: A machine/OS-specific host identifier, unique in the domain. If you know the number of machines in the network ahead of time, the size of the **hostId** field can be reduced. For example, if only two machines are connected in the network, then this field can be reduced to two bits (the identifiers would be '1' and '2', in binary '01' and '11'). The range of host IDs that can be used in the WireProtocolQosPolicy is therefore limited by this value.
- **appId**: A participant-specific identifier, unique within the scope of the hostId. If you know the number of participants in the domain ahead of time, the size of the appId field can be reduced. For example, if only four participants are in the domain, then this field can be reduced to three bits. The range of **appId**'s that can be used in the WireProtocolQosPolicy is therefore limited by this value.
- **instanceId**: An instance-specific identifier of the *DomainParticipant* that, together with the appId, is unique within the scope of the hostId. This identifier is increased each time the participant is recreated in the application. Since most applications create only one *DomainParticipant*, this field can usually be eliminated—in which case a value of 1 is assumed.

The **reduce_guidPrefix** value is expressed as three comma-separated integers between 0 and 32. The values specify the number of bits to use for each identifier (hostId, appId and instanceId).

In the example below, the hostId will use 5 bits (because in our example network there will be 32 machines), the appId field will use 6 bits (because there will be 64 participants in the domain), and the instanceId field will be eliminated. So 11 bits will be used. Then the LBRTPS transport will reduce the **guidPrefix** field 11 bits into 2 bytes.

```
<element>
    <name>
         dds.transport.lbrtps.rtps_header.reduce_guidPrefix
    </name>
    <value>5,6,0 </value>
</element>
```

Important Notes:

1. An integral number of bytes is sent by the plugin for the GUID prefix. So the plugin will round up to the next byte when the total number of bits for the IDs add up to less than an integral number of bytes. For example, a **reduce_guidPrefix** value of 3,2,0 requires 5 bits, so 1 byte will be sent; a value of 4,4,2 requires 10 bits so 2 bytes will be sent; etc.
2. Setting any of the **reduce_guidPrefix** fields to 0 will behave as expected on the sending side (no bits are used to send that ID) but on the receiving side, the plugin will use/assume an ID value of 1 for any IDs that were not sent. Thus, when setting 0 bits for an ID field in the **reduce_guidPrefix**, you must use the WireProtocolQosPolicy to set the corresponding ID to a value of 1 in the local participant.
3. When using this property to reduce the number of bits used to encode an ID, the actual ID (host, app, instance) should be a value that can be represented by the reduced bits. If a full ID is used, aliasing of two full ID values to the same reduced ID value may occur since a bit mask is used to convert a full ID to the reduced ID. This could lead to two different participants having the same reduced GUID prefix. For example, using '2,0,0' for the **reduce_guidPrefix** property (2 bits to encode the host_id and no bits for the app ID or instance ID) and setting the **rtps_host_id** in the WireProtocolQosPolicy to 3 (0011 in binary) for one participant and 7 (0111 binary) for another participant will cause the reduced **rtps_host_id**'s of both participants to be the same value of 3 (0011 binary). In this situation, discovery will not complete.
4. A **reduce_guidPrefix** value of '0,0,0' is not valid. As noted in point 2 above, a **reduce_guidPrefix** of '0,0,0' will be interpreted as a GUID prefix of host ID = 1, app ID = 1 and instance ID = 1. Unfortunately, this would lead to all participants having the same GUID prefix, since a GUID prefix of host ID = 0, app ID = 0 and instance ID = 0 is not allowed. As a side effect, the GUID prefix cannot be reduced to 0 bytes (1-byte minimum used to send the GUID prefix).
5. When setting the WireProtocolQosPolicy of a participant, a value of 0 for an ID is equivalent to setting the value to DDS_RTPS_AUTO_ID. So when using 0 for an ID, DDS will automatically set the value of the ID to some value other than 0.

Chapter 5 Compression Real-Time Publish Subscribe Transport Plugin

Real-Time Publish Subscribe (RTPS) is the communication protocol used by the Data Distribution Service (DDS) interoperability protocol. *Connext DDS* uses RTPS packages to send data over the network.

The Compression Real-Time Publish Subscribe (ZRTPS) transport plugin reduces the size of the RTPS packages sent over the network by *Connext DDS*.

You can configure the ZRTPS transport plugin to use one of the following algorithms to compress all RTPS packages:

- **Zlib**: This compression library is an abstraction of the DEFLATE compression algorithm used in the gzip file compression program. This is free software, distributed under the ZLIB license.

  Windows users: **zlib1.dll** is included in the **<NDDSHOME>\lib\<*architecture*>** directory (where <*architecture*> is one of the supported architecture strings listed in the *Release Notes*).

  Linux users: **libzlib1.so** is likely already installed on your system. If you need to get a Zlib library, please see http://zlib.net for information on how to obtain this library for your platform.

- **Bzip2**: This compression library contains the bzip2 compression algorithm. This algorithm is a lossless data compression algorithm. bzip2 compresses most files more effectively than the older LZW and Deflate compression algorithms, but is considerably slower (~12 times vs. Deflate on typical data). This is free software, distributed under the BSD license.

- **External compression library**: You can add your own compression library and configure the ZRTPS transport plugin to use it.

The transport also supports an automatic mode, which will select from the available algorithms the one that results in the smallest compressed package. While this automatic mode assures the best size reduction, it is slower and uses more memory.

The ZRTPS transport plugin can apply different compression algorithms, depending on each RTPS package's size (small, medium, and large—these sizes are also user-configurable).

The ZRTPS transport works over another transport. It cannot send data over a network by itself. It can work over the *Connext DDS* UDPv4 transport, or a custom transport created using the *Connext DDS* Transports API. You can configure *Connext DDS* to use the ZRTPS transport via the QoS profiles XML; see Configuring the ZRTPS Transport (Section 5.2).

Note: You must link with the *dynamic* version of the *Connext DDS* core libraries. See the *RTI Connext DDS Core Libraries Platform Notes* for details.

5.1 Transport-Related Limitations

The following are known transport-interaction limitations when using the ZRTPS transport plugin:

- Using LBRTPS *and* ZRTPS: See Using ZRTPS as a Subtransport (Section 4.2.1.2).
- Neither Shared Memory (SHMEM) nor UDPv6 may be used as subtransports.
- The UDPv4 transport may not be used simultaneously as a transport and a ZRTPS subtransport.

5.2 Configuring the ZRTPS Transport

This section describes how to configure the properties for the ZRTPS Transport plugin in the XML QoS Profile file used by *Connext DDS* (such as **USER_QOS_PROFILES.XML**), or in the PropertyQosPolicy for your application's *DomainParticipant*.

Before we describe the name/value pairs that can be used in the Property QoS policy to configure the ZRTPS transport, let's review an example. The first step is to disable the built-in transports by configuring the TransportBuiltin QoS policy with the mask **MASK_NONE**. Then the name/value pairs in the Property QoS policy are set up to load and configure the ZRTPS transport plugin.

```
<qos_library name="Property_Library">
   <qos_profile name="Property_Profile">
        <participant_qos>
             ...
```

Disable built-in transports—>

```
             <transport_builtin>
                  <mask>MASK_NONE</mask>
             </transport_builtin>
             <property>
                  <value>
                       <element>
                            <name>dds.transport.load_plugins</name>
                            <value>dds.transport.zrtps</value>
                       </element>
                       <element>
                            <name>dds.transport.zrtps.library</name>
                            <value>rtizrtps</value>
                       </element>
                       <element>
                            <name>dds.transport.zrtps.create_function</
name>
                            <value>ZRTPS_Transport_create_plugin</value>
                       </element>
                       <element>
                            <name>dds.transport.zrtps.subtransport</
name>
                            <value>UDPv4</value>
                       </element>
                      <element>
                            <name>dds.transport.zrtps.aliases</name>
                            <value>zrtps.udpv4</value>
                       </element>
                       <element>
                            <name>dds.transport.zrtps.UDPv4.multicast_en-
abled
```

```
                                    </name>
                                    <value>1</value>
                                </element>
                                <element>
                                  <name>dds.transport.zrtps.compression_library</
name>
                                  <value>AUTOMATIC_COMPRESSION</value>
                                </element>
                                <element>
                                    <name>dds.transport.zrtps.compres-
sion_level</name>
                                    <value>9</value>
                                </element>
                                ...
                            </value>
                        </property>
                        ...
                    </participant_qos>
                </qos_profile>
            </qos_library>
```

When using the above QoS profile, *Connext DDS* will load the ZRTPS transport plugin from the library, **rtizrtps.dll** on Windows systems, or **rtizrtps.so** on Linux systems and call the function **ZRTPS_Transport_create_plugin** to create the ZRTPS transport.

The ZRTPS transport is designed to work over the *Connext DDS* UDPv4 transport. The automatic mode described previously is set in the ZRTPS transport and the compression level for all compression algorithms is set to 9. The ZRTPS transport will be registered in *Connext DDS* with the participant that uses this QoS profile.

Note: *Connext DDS does not assign initial peers to the ZRTPS transport plugin*. You can set the initial peers with NDDS_DISCOVERY_PEERS, as described in the chapter on *Discovery* in the *RTI Connext DDS Core Libraries User's Manual*. The ZRTPS transport plugin example uses the NDDS_DISCOVERY_PEERS file to set the multicast address, zrtps.udpv4://239.255.0.1.

Table 5.1 **Configuration Properties for ZRTPS Plugin**

Property Name	Property Value and Description

dds.transport.load_plugins	Required.
	Comma-separated strings indicating the prefix names of all plugins to be loaded by *Connext DDS*.
	Set the value to **dds.transport.zrtps**.
	Example:
	`<element>`
	`  <name>dds.transport.load_plugins</name>`
	`  <value>dds.transport.zrtps</value>`
	`</element>`

Table 5.1 **Configuration Properties for ZRTPS Plugin**

Property Name	Property Value and Description

dds.transport.zrtps.library	Required.
	The name of the dynamic library that contains the ZRTPS transport plugin implementation. This library must be in the path during run time for use by *Connext DDS*.
	The value must be **rtizrtps**.
	Example:
	`<element>`
	`  <name>dds.transport.zrtps.library</name>`
	`  <value>rtizrtps</value>`
	`</element>`

dds.transport.zrtps.create_function	Required.
	The name of the function that will be called by *Connext DDS* to create an instance of the ZRPTS transport. The function must have the prototype NDDS_Transport_create_plugin.
	Must be set to **ZRTPS_Transport_create_plugin**.
	Example:
	`<element>`
	`  <name>dds.transport.zrtps.create_plugin</name>`
	`  <value>ZRTPS_Transport_create_plugin</value>`
	`</element>`

dds.transport.zrtps.subtransport	Required.
	Name of the plugin to be loaded by the ZRTPS transport. The ZRTPS transport will work over this loaded plugin. The value can be **UDPv4**, **lbst**, or a user-specified string; see Configuring the ZRTPS Transport Plugin's 'Subtransport' Property (Section 5.2.1).
	Example:
	`<element>`
	`  <name>dds.transport.zrtps.subtransport</name>`
	`  <value>UDPv4</value>`
	`</element>`

dds.transport.zrtps.aliases	Required.
	Aliases used to register the ZRTPS transport plugin with the *DomainParticipant*. The transport must have been created by the dds.transport.zrtps.create_function. Aliases should be specified as a comma-separated string, with each comma delimiting an alias.
	Example:
	`<element>`
	`  <name>dds.transport.zrtps.aliases</name>`
	`  <value>zrtps.udpv4</value>`
	`</element>`

Table 5.1 **Configuration Properties for ZRTPS Plugin**

Property Name	Property Value and Description
The following properties are optional.	

dds.transport.zrtps.compression_level	Defines the compression level that the compression algorithm will use for all RTPS packages. In automatic mode (see dds.transport.zrtps.compression_library), all compression algorithms will use this level.
	The value must be an integer between 1 and 9 (inclusive). A lower value will result in less time spent doing the compression; a higher value may result in a higher compression percentage (smaller compressed output).
	Example:
	`<element>`
	`  <name>dds.transport.zrtps.compression_level</name>`
	`  <value>9</value>`
	`</element>`

dds.transport.zrtps.
compression_level.small_packets
dds.transport.zrtps.
compression_level.medium_packets
dds.transport.zrtps.
compression_level.large_packets
	Defines the compression level that the compression algorithm will use for small/medium/large RTPS packages. In automatic mode (see dds.transport.zrtps.compression_library), all compression algorithms will use this level.
	The value must be an integer between 1 and 9 (inclusive). A lower value means more speed compression; a higher value means more size reduction.
	Example:
	`<element>`
	`  <name>`
	`dds.transport.zrtps.compression_level.small_packets`
	`  </name>`
	`  <value>9</value>`
	`</element>`
	`<element>`
	`  <name>`
	`dds.transport.zrtps.compression_level.medium_packets`
	`  </name>`
	`  <value>9</value>`
	`</element>`
	`<element>`
	`  <name>`
	`dds.transport.zrtps.compression_level.large_packets`
	`  </name>`
	`  <value>9</value>`
	`</element>`

Table 5.1 **Configuration Properties for ZRTPS Plugin**

Property Name	Property Value and Description

dds.transport.zrtps.compression_library	Specifies the compression algorithm to be used by the ZRTPS transport for all RTPS packages. These compression algorithms are in external libraries. The ZRTPS transport only uses the libraries whose algorithms will be used. The required libraries must be in the path during run time so the ZRTPS transport can find them.
	There are several compression algorithms that can be used to compress RTPS packages:
	• ZLIB_COMPRESSION: Use the Zlib algorithm from the library **zlib1.dll**.
	• BZIP2_COMPRESSION: Use the Bzip2 algorithm from the library **bzip2.dll**.
	• EXTERNAL_COMPRESSION: Use an external compression library defined in the property dds.transport.zrtps.external_library. See Configuring the External Compression Library (Section 5.2.2).
	• AUTOMATIC_COMPRESSION: Compress RTPS packages with all previous compression algorithms and send the smallest package.
	Example:
	`<element>`
	`<name>`
	`dds.transport.zrtps.compression_library`
	`</name>`
	`<value>AUTOMATIC_COMPRESSION</value>`
	`</element>`

dds.transport.zrtps.
compression_library.small_packets
dds.transport.zrtps.
compression_library.medium_packets
dds.transport.zrtps.
compression_library.large_packets
	Specifies the compression algorithm to be used for small/medium/large RTPS packages. These compression algorithms are in external libraries. The ZRTPS transport only uses the libraries whose algorithms will be used. Required libraries must be in the path during run time so the ZRTPS transport can find them.
	See dds.transport.zrtps.compression_library.
	Example:
	`<element>`
	`<name>`
	`dds.transport.zrtps.compression_library.small_packets`
	`</name>`
	`<value>ZLIB_COMPRESSION</value>`
	`</element>`
	
	`<element>`
	`<name>`
	`dds.transport.zrtps.compression_library.medium_packets`
	`</name>`
	`<value>ZLIB_COMPRESSION</value>`
	`</element>`
	
	`<element>`
	`<name>`
	`dds.transport.zrtps.compression_library.large_packets`
	`</name>`
	`<value>ZLIB_COMPRESSION</value>`
	`</element>`

Table 5.1 **Configuration Properties for ZRTPS Plugin**

Property Name	Property Value and Description

dds.transport.zrtps.external_library	Sets the name of a user's external library that is to be located and used by the ZRTPS transport plugin. See Configuring the External Compression Library (Section 5.2.2).
	Example:
	`<element>`
	`  <name>dds.transport.zrtps.external_library</name>`
	`  <value>external_library.dll</value>`
	`</element>`

dds.transport.zrtps.low_mark,
dds.transport.zrtps.high_mark
	Specifies the size for small, medium, and large RTPS packages. RTPS packages whose size is <= low_mark are considered small packages. Package sizes > low_mark and <= high_mark are considered medium packages. Package sizes > high_mark are considered large packages.
	The value for each property is an integer representing a number of bytes.
	Example:
	`<element>`
	`  <name>dds.transport.zrtps.low_mark</name>`
	`  <value>128</value>`
	`</element>`
	
	`<element>`
	`  <name>dds.transport.zrtps.high_mark</name>`
	`  <value>512</value>`
	`</element>`

dds.transport.zrtps.verbosity	The verbosity for the plugin, for debugging purposes.
	• -1: Silent
	• 0: Exceptions only (default)
	• 1: Warnings
	• 2 and up: Debug
	Example:
	`<element>`
	`  <name>dds.transport.zrtps.verbosity>/name>`
	`  <value>0</value>`
	`</element>`

5.2.1 Configuring the ZRTPS Transport Plugin's 'Subtransport' Property

The required property, **dds.transport.zrtps.subtransport**, specifies the plugin to be loaded by the ZRTPS transport. Supported subtransports are **UDPv4**, **lbst**, or a user-specified transport, as described in the following sections. *Connext DDS* builtin transports other than UDPv4 are not currently supported as subtransports.

Note: The LBRTPS Transport Plugin cannot be used as a subtransport to ZRTPS. If you want to use both plugins, see Using ZRTPS as a Subtransport (Section 4.2.1.2).

5.2.1.1 Using UDPv4 as a Subtransport

To load the *Connext DDS* UDPv4 built-in transport, use the value **UDPv4**. If you want the UDPv4 transport to be created with multicast support, also set **dds.transport.zrtps.UDPv4.multicast_enabled** to 1.

For example:

```
<element>
```

```
    <name>dds.transport.zrtps.subtransport</name>
    <value>UDPv4</value>
</element>
<element>
    <name>dds.transport.zrtps.UDPv4.multicast_enabled</name>
    <value>1</value>
</element>
```

5.2.1.2 Using LBST as a Subtransport

To load the LBST simulation transport described in Chapter 6: Limited Bandwidth Simulation Plugin, use the value **lbst**. You will also need to set additional properties to configure the simulation plugin:

```
<element>
    <name>dds.transport.zrtps.subtransport</name>
    <value>lbst</value>
</element>
<element>
    <name>dds.transport.zrtps.lbst.library</name>
    <value>rtilbst</value>
</element>
<element>
    <name>dds.transport.zrtps.lbst.create_function</name>
    <value>LBS_Transport_create_plugin</value>
</element>
<element>
    <name>dds.transport.zrtps.lbst.aliases</name>
    <value>lbst.udpv4</value>
</element>
<element>
    <name>dds.transport.zrtps.lbst.subtransport</name>
    <value>UDPv4</value>
</element>
<element>
    <name>dds.transport.zrtps.lbst.UDPv4.multicast_enabled</name>
    <value>1</value>
</element>
<element>
    <name>dds.transport.zrtps.lbst.lbsm.address</name>
    <value>127.0.0.1</value>
</element>
<element>
    <name>dds.transport.zrtps.lbst.lbsm.port</name>
    <value>5012</value>
</element>
<element>
    <name>dds.transport.zrtps.lbst.information_port</name>
    <value>5004</value>
</element>
<element>
    <name>dds.transport.zrtps.lbst.events_port</name>
    <value>5003</value>
</element>
<element>
    <name>dds.transport.zrtps.lbst.identification</name>
    <value>PARTICIPANT_IDENTIFICATION</value>
</element>
```

5.2.1.3 Using a User-Specified Subtransport

To load a user-provided transport plugin, provide a value for **dds.transport.zrtps.subtransport**, then use that same value as a prefix to define these two additional properties:

- dds.transport.zrtps.<*prefix*>.library
- dds.transport.zrtps.<*prefix*>.create_function

For example, to specify the value, **testplugin**:

```
<element>
    <name>dds.transport.zrtps.subtransport</name>
    <value>testplugin</value>
</element>
<element>
    <name>dds.transport.zrtps.testplugin.library</name>
    <value>testplugin</value>
</element>
<element>
    <name>dds.transport.zrtps.testplugin.create_function</name>
    <value>TestPlugin_Transport_create_plugin</value>
</element>
```

5.2.2 Configuring the External Compression Library

The ZRTPS transport plugin loads the external compression library defined in the QoS profile (see dds.transport.zrtps.compression_library). The transport will try to detect the following three functions, which must be implemented in the external library:

- ZRTPS_Transport_external_calculate_length (Section 5.2.2.1)
- ZRTPS_Transport_external_compress (Section 5.2.2.2)
- ZRTPS_Transport_external_uncompress (Section 5.2.2.3)

5.2.2.1 ZRTPS_Transport_external_calculate_length

This function is called before compressing the data.

```
int ZRTPS_Transport_external_calculate_length(int data_length);
```

Parameters:

- **data_length** is the size of the data that will be compressed.

Returns: Maximum size of the buffer that the external library needs when the compression function is called.

5.2.2.2 ZRTPS_Transport_external_compress

This function is called when data has to be compressed.

```
int ZRTPS_Transport_external_compress(char *dst,
                                      int *dst_length,
                                      const char *src,
                                      int src_length,
                                      int compression_level);
```

Parameters:

- **dst** and **src** are pointers to the destination and source buffers, respectively.
- **dst_length** is the size of the destination buffer; this size is returned by **ZRTPS_Transport_external_calculate_length.dst_length** must return the actual size of the compressed data.

- **compression_level** is a value between 1 and 9 (inclusive). Some compression libraries use this and others do not.

Returns: 0 if successful, -1 if unsuccessful.

5.2.2.3 ZRTPS_Transport_external_uncompress

This function is called to uncompress data.

```
int ZRTPS_Transport_external_uncompress(char *dst,
                                        int *dst_length,
                                        const char *src,
                                        int src_length);
```

Parameters:

- **dst** and **src** are pointers to the destination and source buffers, respectively.
- **dst_length** must be the size of the entire destination buffer.
- **src_length** is the size of the compressed data in the source buffer.

Returns: Actual size of the uncompressed data.

Chapter 6 Limited Bandwidth Simulation Plugin

The Limited Bandwidth Simulation Plugin (LBSP) provides a way to simulate a Limited bandwidth, shared communication channel. It includes two components:

- The Limited Bandwidth Simulation Transport (LBST) is a transport to be used with *Connext DDS*. Each participant to be simulated on the shared communication channel must use this transport. This transport sends information about the RTPS packages to the LBSM (described below). See Configuring the Limited Bandwidth Simulation Transport (LBST) (Section 6.2)
- The Limited Bandwidth Simulation Manager (LBSM) is an external library. This library centralizes information about the RTPS packages that are sent and is responsible for the simulation. It receives information about all LBSTs. An example application that uses this library is provided. See Configuring the Limited Bandwidth Simulation Manager (LBSM) (Section 6.3).

Note: You must link with the *dynamic* version of the *Connext DDS* libraries. See the *RTI Connext DDS Core Libraries Platform Notes* for details.

6.1 Key Concepts

Bandwidth: Bandwidth specifies how much information can be transmitted per unit time. It is measured in bits per second.

Buffer: Devices that share a common channel and have to wait until it is free prior to transmitting usually have an internal buffer where the information is stored prior to being sent. Theoretical models usually do not take into account the existence of this buffer, but applications developers have to deal with it and the constraints it imposes. The following parameters model buffer behavior:

- Size, measured in bytes.
- **Overflow**, which controls what to do with the information when the buffer is full and an application tries to write. If there is an overflow, the information will be silently discarded. If there is no overflow, the information will be kept, regardless of buffer size, and the buffer has no effect on packet losses.

Advance channel layer: On a shared channel, most packet losses and delays are caused by collisions. Collisions happen when two or more applications try to transmit at the same time. To avoid, minimize, or reduce the impact of collisions, communication systems have a layer known as Medium Access Control (MAC), which can be seen as a set of rules, criteria and algorithms that all the systems using the channel must follow.

MAC algorithms and procedures are oriented to optimize channel use. Of course, the terms "optimize" and "use" are subject of interpretation and many different algorithms have been developed over the years. Some are designed to optimize throughput while others optimize latency, or both, based on the physical channel characteristics and traffic.

Instead of simulating several specific MAC implementations[1], the Limited Bandwidth Simulation Transport (LBST) has a generic MAC layer with parameters that let you simulate the effects of most of them.

This release includes simulation of generic multiple access algorithms, those where any station can transmit data at any time. Future releases may include TDMA and/or other collision-free access control methods.

Our generic MAC Layer puts information on the channel in chunks or frames of a determined maximum size. It may wait for a time before and/or after putting the data on the channel. We call these wait times *guards* that model the effect of different MAC algorithms on channel usage.

Frame size: Maximum amount of information that can be sent on the channel at once, in bytes.

Pre-transmission guard: Time, in milliseconds, that a station may wait before transmitting the real data. It captures the effect of MAC procedures for getting the channel (e.g., reservation) or other device-related delays.

Post-transmission guard: Time, in milliseconds, that a station may wait before transmitting the real data. It captures the effect of unknown MAC procedures for freeing the channel, ensuring that data have been transmitted correctly, or other device-related delays.

Burst traffic: Some advanced MAC protocols let the channel owner send several frames, avoiding channel contention on each frame and increasing throughput when occupation is high.

During a burst, pre- and post-transmission guards may have different values for frames following the first.

- **Burst length:** Maximum number of frames that can be transmitted in sequence.
- **Burst frame size:** Maximum amount of information that can be sent on the channel at once, in bytes.
- Burst pre-transmission guard.
- Burst post-transmission guard.

6.2 Configuring the Limited Bandwidth Simulation Transport (LBST)

The LBST must be created using the *Connext DDS* Transport API; for more information, please see the chapter on *Transport Plugins* in the *RTI Connext DDS Core Libraries User's Manual.*

Before we describe the name/value pairs that can be used in the Property QoS policy to configure the LBS transport, let's review an example. The first step is to disable the built-in transports by configuring the

1. Specific MAC simulations may be added in the future. Please, contact us for specific MAC algorithm simulation cost estimation.

TransportBuiltin QoS policy with the mask **MASK_NONE**. Then the name/value pairs in the Property QoS policy are set up to load and configure the LBS transport plugin.

```
<qos_library name="Property_Library">
    <qos_profile name="Property_Profile">
        <participant_qos>
            ...
            <transport_builtin>
                <mask>MASK_NONE</mask>
            </transport_builtin>
            <property>
                <value>
                    <element>
                        <name>dds.transport.load_plugins</name>
                        <value>dds.transport.lbst</value>
                    </element>
                    <element>
                        <name>dds.transport.lbst.library</name>
                        <value>rtilbst</value>
                    </element>
                    <element>
                        <name>dds.transport.lbst.create_function
                        </name>
                        <value>LBS_Transport_create_plugin</value>
                    </element>
                    <element>
                        <name>dds.transport.lbst.subtransport</name>
                        <value>UDPv4</value>
                    </element>
                    <element>
                        <name>dds.transport.lbst.aliases</name>
                        <value>lbst.udpv4</value>
                    </element>
                    <element>
                       <name>dds.transport.lbst.UDPv4.multicast_enabled
                       </name>
                       <value>1</value>
                    </element>
                    <element>
                        <name>dds.transport.lbst.lbsm.address</name>
                        <value>192.168.1.21</value>
                    </element>
                    <element>
                        <name>dds.transport.lbst.lbsm.port</name>
                        <value>5002</value>
                    </element>
                    <element>
                        <name>dds.transport.lbst.identification</name>
                        <value>PARTICIPANT_IDENTIFICATION</value>
                    </element>
                    ...
                </value>
            </property>
            ...
        </participant_qos>
    </qos_profile>
</qos_library>
```

Disable built-in transports—>

When using the above QoS profile, *Connext DDS* will load the LBST plugin from the library **lbst.dll** and call the function **LBS_Transport_create_plugin**, which will create the LBST. The LBST is designed to work over the *Connext DDS* UDPv4 transport. The LBST will connect with the LBSM on the IP address 192.168.1.21 and UDP port 5002. The LBSM knows the LBST by the participant ID. The LBST will be registered in *Connext DDS* with the participant that uses this QoS profile.

Note: *Connext DDS* does not assign initial peers to the LBS transport plugin. You can set the initial peers with NDDS_DISCOVERY_PEERS, as described in the chapter on *Discovery* in the *RTI Connext DDS Core Libraries User's Manual*. The LBS transport plugin examples use an NDDS_DISCOVERY_PEERS file to set the multicast address, lbst.udpv4://239.255.0.1.

The following table describes the name/value pairs that you can use to configure the LBST.

Table 6.1 **Configuration Properties for LBST Plugin**

Property Name	Property Value and Description

dds.transport.
load_plugins
	Required.
	Comma-separated strings indicating the prefix names of all plugins to be loaded by *Connext DDS*.
	Must be set to **dds.transport.lbst**
	Example:
	`<element>`
	`<name>dds.transport.load_plugins</name>`
	`<value>dds.transport.lbst</value>`
	`</element>`

dds.transport.lbst.
library
	Required.
	The name of the dynamic library that contains the LBST transport plugin implementation. This library must be in the path during run time so it can be used by *Connext DDS*.
	Set this value to **rtilbst**.
	Example:
	`<element>`
	`<name>dds.transport.lbst.library</name>`
	`<value>rtilbst</value>`
	`</element>`

dds.transport.lbst.
create_function
	Required.
	The name of a function, with the prototype of **NDDS_Transport_create_plugin**, that will be called by *Connext DDS* to create an instance of LBST transport.
	Set this value to **LBS_Transport_create_plugin**.
	Example:
	`<element>`
	`<name>dds.transport.lbst.create_plugin</name>`
	`<value>LBS_Transport_create_plugin</value>`
	`</element>`

dds.transport.lbst. subtransport	Required. Name of the plugin that will be loaded by LBST transport. The LBST transport will work over this loaded plugin. The value can be **UDPv4** or a user-specified string; see Configuring the LBST Subtransport (Section 6.2.1).

Table 6.1 **Configuration Properties for LBST Plugin**

Property Name	Property Value and Description

dds.transport.lbst.aliases	Required.
	Aliases used to register the LBST transport plugin with the DomainParticipant. The transport must have been created by the **dds.transport.lbst.create_function**. Aliases should be specified as a comma-separated string, with each comma delimiting an alias.
	An example alias to LBST transport working over a *Connext DDS* UDPv4 transport: lbst.udpv4
	Example:
	`<element>`
	`  <name>dds.transport.lbst.aliases</name>`
	`  <value>lbst.udpv4</value>`
	`</element>`

The following properties are optional and appear in alphabetical order.	

dds.transport.lbst.events_port	UDP port where LBST receives information from LBSM application.
	The value is a port number (0 to 65535).
	Example:
	`<element>`
	`  <name>dds.transport.lbst.events_port</name>`
	`  <value>5021</value>`
	`</element>`

dds.transport.lbst.identification	The LBSM application has to identify each LBST connected to it. This property specifies what kind of identification will be used by LBSM.
	There are two kinds of identification:
	• IP_IDENTIFICATION: Each LBST will be known by its host. This kind of identification can only be used when each DomainParticipant is on a different host.
	• PARTICIPANT_IDENTIFICATION: Each LBST will be known by the DomainParticipant ID.
	Example:
	`<element>`
	`  <name>dds.transport.lbst.identification </name>`
	`  <value>PARTICIPANT_IDENTIFICATION</value>`
	`</element>`

dds.transport.lbst.information_port	UDP port where LBST sends information to LBSM application.
	The value is a port number (0 to 65535).
	Example:
	`<element>`
	`  <name>dds.transport.lbst.information_port</name>`
	`  <value>5011</value>`
	`</element>`

dds.transport.lbst.lbsm.address	IP address where the LBSM application is running.
	IPv4 format: X.X.X.X
	Example:
	`<element>`
	`  <name>dds.transport.lbst.lbsm.address</name>`
	`  <value>192.168.1.21</value>`
	`</element>`

Table 6.1 **Configuration Properties for LBST Plugin**

Property Name	Property Value and Description
dds.transport.lbst. lbsm.port	UDP port where the LBSM application is listening. The value is a port number (0 to 65535). Example: <element> <name>dds.transport.lbst.lbsm.port</name> <value>5002</value> </element>
dds.transport.lbst. verbosity	The verbosity for the plugin, for debugging purposes. • -1: Silent • 0: Exceptions only (default) • 1: Warnings • 2 and up: Debug Example: <element> <name>dds.transport.lbst.verbosity>/name> <value>0</value> </element>

6.2.1 Configuring the LBST Subtransport

The required property, **dds.transport.lbst.subtransport**, specifies the plugin to be loaded by the LBS transport. The value can be **UDPv4** or a user-specified value, as described below.

- To load the *Connext DDS* UDPv4 built-in transport, use the value **UDPv4**. If you want the UDPv4 transport to be created with multicast support, also set **dds.transport.lbst.UDPv4.multicast_enabled** to 1.

  To use the UDPv4 transport and enable multicast:

```
<element>
        <name>dds.transport.lbst.subtransport</name>
        <value>UDPv4</value>
</element>
<element>
        <name>dds.transport.lbst.UDPv4.multicast_enabled</name>
        <value>1</value>
</element>
```

- To load a user-provided transport plugin, provide a value for **dds.transport.lbst.subtransport,** and then use that same value like a prefix to define these two additional properties:
  - dds.transport.lbst.*prefix*.library
  - dds.transport.lbst.*prefix*.create_function

  For example, to specify the value, **testplugin**:

```
<element>
        <name>dds.transport.lbst.subtransport</name>
        <value>testplugin</value>
</element>
<element>
        <name>dds.transport.lbst.testplugin.library</name>
        <value>testplugin.dll</value>
```

```
    </element>
    <element>
            <name>dds.transport.lbst.testplugin.create_function</name>
            <value>TestPlugin_Transport_create_function</value>
    </element>
```

6.3 Configuring the Limited Bandwidth Simulation Manager (LBSM)

This section describes the XML tags you can use to configure the LBSM and the shared communication channel that will be simulated. First, let's see an example:

```
<LowBandwidthSimulationManagerProfile>
    <lbsm>
         <listener_port>5012</listener_port>
         <event_port>5011</event_port>
    </lbsm>
    <station>
         <overflow>true</overflow>
         <buffer_max_size>102400</buffer_max_size>
    </station>
    <channel type="ADVANCE_CHANNEL_LAYER">
         <bandwidth>1200</bandwidth>
         <packet_loss>3</packet_loss>
         <propagation_delay>0</propagation_delay>
         <frame_size>410</frame_size>
         <burst_frame_size>530</burst_frame_size>
         <normal_pretransmission_guard>400</normal_pretransmission_guard>
        <normal_posttransmission_guard>200</normal_posttransmission_guard>
         <burst_pretransmission_guard>0</burst_pretransmission_guard>
         <burst_posttransmission_guard>200</burst_posttransmission_guard>
         <burst_length>3</burst_length>
         <retransmissions>6</retransmissions>
    </channel>
</LowBandwidthSimulationManagerProfile>
```

Table 6.2 through Table 6.5 describe the tags supported within the root tag, **<LowBandwidthSimulationManagerProfile>**. All tags are optional unless otherwise noted and cannot be repeated (that is, you can only have one of each tag).

6.4 Running the Example Application

Before your application can use the LBST plugin, the LBSM has to be executed. A simple application to run the LBSM library is provided. To run the example, enter:

```
rtilbsimmanager [XML configuration file]
```

It accepts a parameter that specifies the URL of an XML configuration file.

Table 6.2 **Top-level Tags for LBSM Configuration**

Tags within <LowBandwidth SimulationManagerProfile>	Description

<channel type>	Required.
	Specifies one of two simulation types:
	• SIMPLE_CHANNEL_LAYER: Simple simulation that simulates bandwidth restrictions, packet loss and propagation delay. There aren't packet collisions.
	• ADVANCE_CHANNEL_LAYER: Complex simulation that was described previously.
	Example/Default:
	`<channel type=" SIMPLE_CHANNEL_LAYER">`
	`...`
	`</channel>`
	See Table 6.3, "Channel Tags"

<lbsm>	Required. Configures the simulation manager application (**rtilbsimmanager.exe**). See Table 6.4, "LBSM Tags"
<station>	Required. Configures simulation attributes that are applied in each *Connext DDS* application. For example, each *Connext DDS* application uses a radio and each radio has a buffer that could be used for overflow. See Table 6.5, "Station Tags"

Table 6.3 **Channel Tags**

Tags within <channel>	Description
The following only apply when <channel type = SIMPLE_CHANNEL_LAYER>	
<bandwidth>	How much data can be transmitted per unit time in bytes per second. Example/Default: `<bandwidth>1200</bandwidth>`
<packet_loss>	Percentage of packets that will be lost in the simulation. The value 3 means that for each 100 packets, 3 will be lost. Example/Default: `<packet_loss>3</packet_loss>`
<propagation_delay>	Communication delay to be applied in the simulation, between when an application sends the packet and the other application receives it, in milliseconds. Example/Default: `<propagation_delay>0</propagation_delay>`
The following only apply when <channel type = ADVANCE_CHANNEL_LAYER>	
<burst_length>	Maximum number of frames that can be transmitted in sequence in burst mode. Example/Default: `<burst_length>3</burst_length>`
<burst_frame_size>	Maximum amount of data that can be sent on the channel at once in a burst frame, in bytes. Example/Default: `<burst_frame_size>530</burst_frame_size>`

<burst_pretransmission_guard>	Time, in milliseconds, that a station may wait before transmitting the real data in burst frame.
	Example/Default:
	`<burst_pretransmission_guard>`
	`0`
	`</burst_pretransmission_guard>`

Table 6.3 **Channel Tags**

Tags within <channel>	Description

<burst_posttransmission_guard>	Time, in milliseconds, that a station may wait before transmitting the real data in burst frame.
	Example/Default:
	`<burst_posttransmission_guard>`
	`200`
	`</burst_posttransmission_guard>`

<frame_size>	Maximum amount of data that can be sent on the channel at once, in bytes. Example/Default: `<frame_size>410</frame_size>`

<normal_posttransmission_guard>	Time, in milliseconds, that a station may wait before transmitting the real data.
	Example/Default:
	`<normal_posttransmission_guard>`
	`200`
	`</normal_posttransmission_guard>`

<normal_pretransmission_guard>	Time, in milliseconds, that a station may wait before transmitting the real data.
	Example/Default:
	`<normal_pretransmission_guard>`
	`400`
	`</normal_pretransmission_guard>`

<retransmissions>	In the case of a collision, number of retries to send the data. Example/Default: `<retransmissions>6</retransmissions>`

Table 6.4 **LBSM Tags**

Tags within <lbsm>	Description
<event_port>	Required. Port used by the LBSM to send information (events) to *Connext DDS* applications that use the LBST plugin. Example: `<event_port>5002</event_port>`
<listener_port>	Required. Port on which the LBSM to receive information from *Connext DDS* applications that use the LBST plugin. Example: `<listener_port>5001</listener_port>`

6.5 Creating Your Own Application

LBSM is a C++ library. Three classes will be used: **LBSMOptions, LowBandwidthSimulationManagerXmlParser** and **LowBandwidthSimulationManager**.

1. Create an **LBSMOptions** object.

```
LBSMOptions *options = new LBSMOptions();
```

Table 6.5 **Station Tags**

Tags within <station>	Description
<overflow>	Required. Each station (LBST) has a buffer where the RTPS packages are stored before being sent. This property controls whether this buffer has unlimited size or a fixed size. See also: <buffer_max_size>. The property value can be either: • true: The buffer has a fixed size and overflow may occur. • false: The buffer has unlimited size. Example/Default: `<overflow>false</overflow>`
<buffer_max_size>	Each station (LBST) has a buffer where the RTPS packages are stored before they are sent. This property specifies the size of this buffer in bytes. It is only used if the <overflow> property is true. Example/Default: `<buffer_max_size>102400</buffer_max_size>` **Note:** Setting this size to 0 means an unlimited buffer size. Therefore if you set <overflow> to true and buffer_max_size to 0, this is equivalent as setting <overflow> to false.

2. The LBSMOptions object contains the configuration of the LBSM and the shared communication channel that will be simulated. When this object is created, the properties are created with default values.

   Use the class, **LowBandwidthSimulationManagerXmlParser**, to fill in the **LBSMOptions** object with the configuration from your XML file.

```
LowBandwidthSimulationManagerXmlParser::parseFile(
                        "Configuration.xml", options);
```

3. Create a **LowBandwidthSimulationManager** object and start it.

```
LowBandwidthSimulationManager *lbsm =
                        new LowBandwidthSimulationManager(options);
lbsm->start();
```

   The main thread can sleep now.

6.6 Troubleshooting Socket-Creation Errors

If you see an error message indicating the transport cannot create sockets, a possible cause is that the ports it uses by default are already in use. In this case, you may need to make configuration changes to use different ports.

The LBSM listener_port must be the same as the dds.transport.lbst.lbsm.port setting in LBST. All of the other ports must be unique and are used for various communications.

To change the port used by LBST:

```
<element>
    <name>dds.transport.lbst.lbsm.port</name>
    <value>5012</value>
</element>
<element>
    <name>dds.transport.lbst.information_port</name>
```

```
    <value>5002</value>
</element>
<element>
    <name>dds.transport.lbst.events_port</name>
    <value>5001</value>
</element>
```

To change the port used by LBSM:

```
<lbsm>
    <listener_port>5012</listener_port>
    <event_port>5011</event_port>
</lbsm>
```