Using Wireshark

with
RTI Connext DDS

Getting Started Guide

Version 1.99.1_RTI520

Printed in U.S.A. First printing.
June 2015.

Trademarks

Copy and Use Restrictions

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/

Table of Contents

Chapter 1 Introduction

Wireshark™ is a network-packet analyzer that supports many network protocols, including *Real-Time Publish-Subscribe (RTPS),* the wire protocol used by *RTI® Connext™ DDS*.

Wireshark can be used to capture and analyze RTPS packets from Connext DDS applications. It supports RTPS 2.1 (and lower) and is specifically tailored to make RTPS packet analysis easier by including:

- A set of predefined filters to quickly select different groups of packets from the RTPS protocol.
- Coloring rules that highlight important RTPS packets. Packets not strictly related to Connext DDS traffic are grayed-out.
- Support to dissect RTI TCP messages. This includes, RTI TCP Control messages as well as RTPS over RTI TCP.

Network packet and traffic analyzers are used during application development and distributed system configuration to monitor the packets over the network. You can use filters to capture specific types of packets, then analyze the captured packets.

This manual will help you use *Wireshark* to analyze RTPS packets. This analysis will give you information on two levels:

- A high-level look at the RTPS submessages that are flowing between your Connext DDS applications.
- A detailed look at the contents of individual packets.

Wireshark's main window makes it easy to see both views.

1.1 Available Documentation

- RTI Wireshark Release Notes describes system requirements, installation instructions, and other important information.
- *RTI Connext DDS Core Libraries User's Manual* provides details on the Connext DDS API and describes how RTPS packets are used by Connext DDS-based applications. In particular, you should review the Discovery chapter. Open **<NDDSHOME>/doc/manuals/connext_dds/RTI_ConnextDDS_CoreLibraries_UsersManual.pdf**, where <NDDSHOME> is where you installed Connext DDS.
- *RTPS Specification*. Please see http://www.omg.org/spec/DDSI/2.1/.
- *Wireshark online help*. There is extensive online help included with Wireshark. Select **Help, Contents** from the menu bar for a detailed user's guide in HTML format.
- *Wireshark User's Guide* describes how to use *Wireshark*'s features. It is not included in the installation, but can be downloaded from *Wireshark's* website (www.wireshark.org/docs). Note that it may pertain to a slightly different version of *Wireshark*.

1.2 Reading Guide

We suggest that you read the documentation in the following order:

- Read this section to become familiar with the system requirements.
- Read the RTI Wireshark Release Notes.
- Follow the steps in Chapter 2 Installation.
- Read Chapter 4 Capturing RTPS Packets for a quick overview of how to capture RTPS packets.
- Read Chapter 5 Analyzing RTPS Packets to learn how to analyze each type of RTPS packet by looking at sample files of captured RTPS packets. During this process, you will need to reference the Real-Time Publish-Subscribe Wire Protocol Specification.
- Read Chapter 6 Practical Uses with RTI Applications for ideas on how to use Wireshark during Connext DDS application development.
- Consult the Wireshark online help and user's guide for information on other features.

1.3 How to Get Support

Technical support for *Wireshark* is provided by RTI; send e-mail to **support@rti.com**.

Wireshark is an open source product. For information about *Wireshark* support, please visit www.wireshark.org.

Chapter 2 Installation

To install *Wireshark*, you need to login as super-user on Linux and OS X systems, or as administrator on Windows systems.

You may also need super-user/administrator access to capture packets. (With normal user access, you may be able to run *Wireshark*, but only view previously saved capture files.)

2.1 Before Installation

If you have *Ethereal* or *Wireshark* already installed, we recommend that you remove them before installing RTI's distribution of *Wireshark*. See 2.6 Uninstalling Wireshark.

2.2 Installing Wireshark on Windows Systems

1. Right-click on the distribution file, **Wireshark-<*version*>-Win32.exe**, and select **Run as Administrator**.

   This will install Wireshark (QT-based GUI), Wireshark Legacy (GTK-based GUI) and Tshark, a terminal-based (non-GUI) version of Wireshark. The default installation path is **C:\Program Files (x86)\Wireshark**.
2. *Wireshark* requires WinPcap 4.1. If WinPcap 4.1 is not already installed, it will be installed with *Wireshark*. If it is already installed, you will be asked if you want to re-install WinPcap or skip the WinPcap installation. You can safely skip re-installing WinPcap.

2.3 Installing Wireshark on Linux (Red Hat) Systems

Install *Wireshark* using the Red Hat Package Manager (RPM):

1. Login as super-user.
2. `cd <location of the distribution file>`

3. `rpm -i wireshark-<version>-<architecture>.rpm`
4. `rpm -i wireshark-gnome<version>-<architecture>.rpm`

For more information on installing RPMs, see http://www.rpm.org. For more information regarding the dependencies and installation, see the *Release Notes*.

2.4 Installing Wireshark on Linux (Debian) Systems

Before Installation:

Make sure you have installed the required packages listed in the Release Notes.

Installation:

The Wireshark package installs the GUI (GTK-based GUI for Ubuntu 12.04 and QT-based GUI for Ubuntu 14.04), while the Tshark package installs a terminal-based (non-GUI) version. For more information on Debian packages, please see http://packages.debian.org.

1. Login as super-user.
2. `cd <location of distribution files>`
3. Install using dpkg the following libraries: wsutil, wiretap, wireshark-data, and wireshark.
4. Install the common package: `dpkg -i wireshark_common-<version>-<architecture>.deb`
5. To install Wireshark (GTK-based GUI), enter:

   ```
   dpkg -i wireshark-gtk_<version>_<architecture>.deb
   ```

6. To install Wireshark (QT-based GUI), enter:

   ```
   dpkg -i wireshark-qt_<version>_-<architecture>.deb
   ```

7. To install Tshark, enter:

   ```
   dpkg -i tshark-<version>-<architecture>.deb
   ```

For more information regarding the dependencies and installation, see the Release Notes.

2.5 Installing Wireshark on OS X Systems

Before Installation:

- Make sure you have *root* privileges.

Installation:

- Double-click the installer and follow the installation process.

2.6 Uninstalling Wireshark

- To uninstall from a Linux (Red Hat) system, while logged in as *root*, enter:

```
rpm -e wireshark
```

- To uninstall from a Linux (Debian) system, while logged in as *root*:

  Uninstall Wireshark:

```
dpkg -r wireshark_common wireshark
```

  Uninstall Tshark:

```
dpkg -r wireshark_common tshark
```

  Uninstall the common package:

```
dpkg -r wireshark wireshark-common
```

- To uninstall from an OS X system, while logged in as *root*, drag the Wireshark .app from /Applications to the bin.
- To uninstall from a Windows system: From the **Start** menu, select **Control Panel, Add/Remove Programs** (or **Programs and Features**), **Wireshark**.

Chapter 3 Starting Wireshark

> To capture packets from the network, you may need to run Wireshark as root/administrator.

- On Windows systems: Use the **Start** menu to select **Wireshark**.
- On Linux systems:

  ```
  # /usr/bin/wireshark &or /usr/bin/wireshark-gtk &
  ```

- On OS X systems: Wireshark will be available in /**Applications**

Alternatively on Linux and OS X systems, to capture with **tshark**, the terminal-based version of Wireshark, run **tshark** from the same paths as above. On Windows systems, you can start **tshark** from **<installation directory>/tshark**.

Chapter 4 Capturing RTPS Packets

This chapter describes how to capture RTPS packets that are sent across a network. After capturing packets, use the information in Chapter 5 Analyzing RTPS Packets to analyze them.

Wireshark will automatically capture all RTPS packets from the wire.

You can create additional filters to refine the scope of your captures. For example, you can create filters to capture packets from specific nodes, addresses, ports, protocols, etc. This chapter provides basic instructions on using capture filters and a few examples. For more information, see the *Wireshark User's Guide* or online documentation.

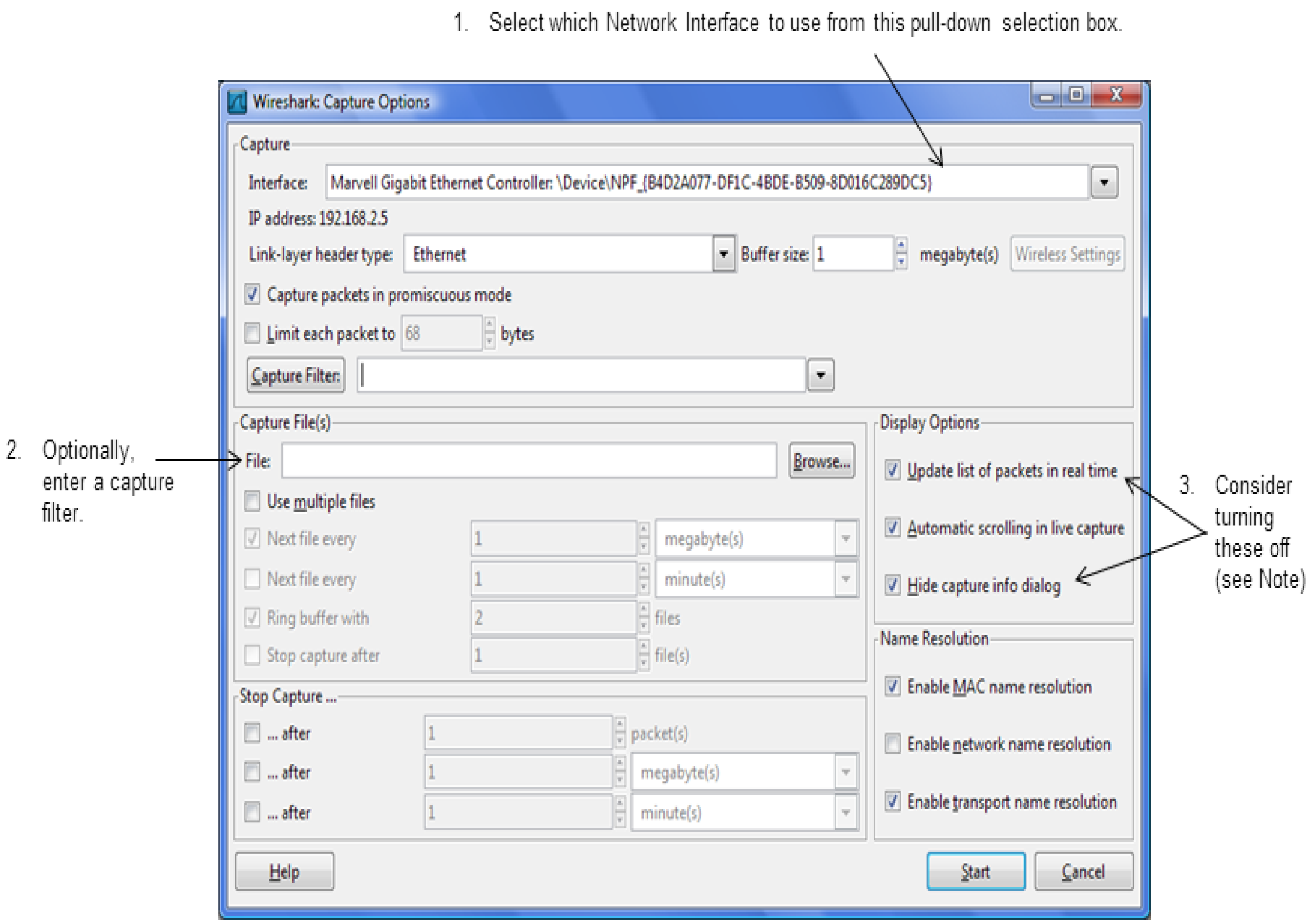
To capture all types of packets while running a Connext DDS application:

1. Login as super-user (on Linux/OSX systems) or administrator (on Windows systems).
2. Start *Wireshark*.
3. Select **Capture**, **Options...** from the menubar. Figure 4.1 Starting a Capture Session shows a sample Capture Options window.

   The defaults in the Capture Option window may very well suit your needs—they will capture all packets sent to the selected interface. Then you can filter the displayed results with a display filter, as described in 5.2.1 Using a Display Filter.

   If you want to change any of the defaults for this window, see the *Wireshark User's Guide* or online help.

Figure 4.1 Starting a Capture Session



Note: Wireshark does not validate capture filter strings as they are entered. It will not alert you about an incorrect expression until after you press the **Capture** button. It may be helpful for you to test your capture filter string with the -f argument to try a capture filter expression. (See **Help, Manual Pages, wireshark**.)

Note:

The check boxes for "Update list of packets in real-time" and “Hide capture info dialog” are selected by default. However, these features can slow down the capture process and increase the chance of missing packets. We recommend deselecting these two check boxes to limit the risk of missing packets.

4. Click **Start** to start the capture session.
5. To stop the capture:

- If “Hide capture info dialog” is selected, click **Stop** on the Capture window or use the  button located on the far right of the main window’s tool bar (you may need to resize the main window to see it).
- If “Hide capture info dialog” is not selected, click **Stop** in the Capture Dialog window.

With the above steps, you will capture *all* the packets that come through your selected interface. Such an indiscriminate capture session may yield hundreds or thousands of packets. While modern computers are amazingly fast, processing each captured packet does take a certain amount of time. Filtering out uninteresting packets can help you squeeze the most out of your computer. Therefore we suggest that you apply a capture filter so that *Wireshark* only captures the type of packets you want to see.

Simply enter a valid capture filter string in the **Capture Filter** box (see Figure 4.1 Starting a Capture Session) before you press **Capture**. Table 4.1 , Example Capture Filters provides some examples.

Table 4.1 Example Capture Filters

To Capture ...	Enter ...
Capture only RTPS	rtps
Only UDP packets	udp
Only RTPS from 10.10.1.92	rtps && ip.src == 10.10.1.92
Only RTPS to 10.10.1.92	rtps && ip.dst == 10.10.1.92
Only RTITCP	rtitcp
Only RTITCP Control Messages (not RTPS data)	rtitcp && !rtps

For more information, see *Wireshark*’s documentation (**Help, Wireshark Online, User’s Guide**).

For help analyzing captured RTPS packets, see Chapter 5 Analyzing RTPS Packets.

Chapter 5 Analyzing RTPS Packets

This section will help you interpret the submessages within captured RTPS packets. There are two levels of analysis that you may be interested in:

- A high-level understanding of what is transpiring during a sequence of captured RTPS packets.

  This section will help you learn to “read” a sequence of packets by walking through the provided sample capture files. You may also find it helpful to review the Discovery section in the *RTI Connext DDS Core Libraries User’s Manual*.

- A more in-depth understanding of an individual packet’s contents.

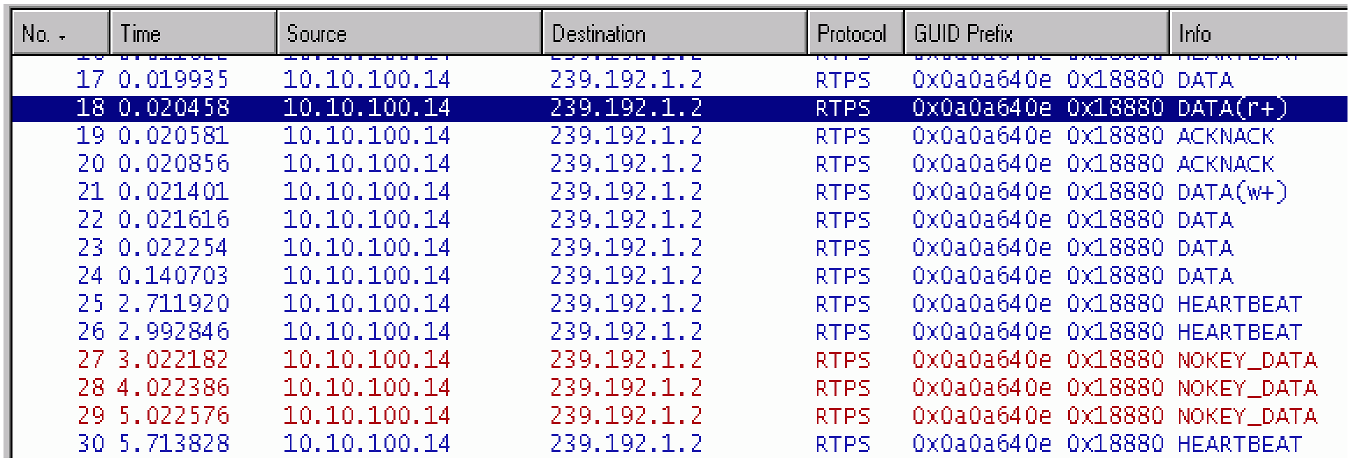
  This section will show you how to display the decoded contents of individual packets. *Wireshark* decodes each RTPS packet and shows you the value for each field in the packet’s structure.

  While the low-level details of a packet’s contents are beyond the scope of this manual, this information is available in the *Real-Time Publish-Subscribe Wire Protocol Specification* (see 1.1 Available Documentation).

This section includes:

5.1 RTPS Submessage Types

Each RTPS packet (message) consists of a header and one or more submessages. When you display captured packets, the Info column (seen in) lists the types of submessages in each packet.



No.	Time	Source	Destination	Protocol	GUID Prefix	Info
17	0.019935	10.10.100.14	239.192.1.2	RTPS	0x0a0a640e 0x18880	DATA
18	0.020458	10.10.100.14	239.192.1.2	RTPS	0x0a0a640e 0x18880	DATA(r+)
19	0.020581	10.10.100.14	239.192.1.2	RTPS	0x0a0a640e 0x18880	ACKNACK
20	0.020856	10.10.100.14	239.192.1.2	RTPS	0x0a0a640e 0x18880	ACKNACK
21	0.021401	10.10.100.14	239.192.1.2	RTPS	0x0a0a640e 0x18880	DATA(w+)
22	0.021616	10.10.100.14	239.192.1.2	RTPS	0x0a0a640e 0x18880	DATA
23	0.022254	10.10.100.14	239.192.1.2	RTPS	0x0a0a640e 0x18880	DATA
24	0.140703	10.10.100.14	239.192.1.2	RTPS	0x0a0a640e 0x18880	DATA
25	2.711920	10.10.100.14	239.192.1.2	RTPS	0x0a0a640e 0x18880	HEARTBEAT
26	2.992846	10.10.100.14	239.192.1.2	RTPS	0x0a0a640e 0x18880	HEARTBEAT
27	3.022182	10.10.100.14	239.192.1.2	RTPS	0x0a0a640e 0x18880	NOKEY_DATA
28	4.022386	10.10.100.14	239.192.1.2	RTPS	0x0a0a640e 0x18880	NOKEY_DATA
29	5.022576	10.10.100.14	239.192.1.2	RTPS	0x0a0a640e 0x18880	NOKEY_DATA
30	5.713828	10.10.100.14	239.192.1.2	RTPS	0x0a0a640e 0x18880	HEARTBEAT

The Info column shows you what submessages are in each packet. The highlighted packet contains a Reader announcement.

Table 5.1 , RTPS 2.x Submessage Types lists the submessages you may see in the Info Column. The details of each type of submessage are described in the *Real-Time Publish-Subscribe Wire Protocol Specification.*

Table 5.1 RTPS 2.x Submessage Types

Submessage Type	Description
ACKNACK	Provides information on the state of a Reader to a Writer.
ACKNACK_BATCH	Provides information on the state of a Reader to a Writer for batched data.
ACKNACK_SESSION	Provides information on the state of a Reader to a multi-channel Writer

Table 5.1 RTPS 2.x Submessage Types

<table>
<tr><th>Submessage Type</th><th>Description</th></tr>
<tr><td>DATA</td><td>
Contains information regarding the value of an application Data-object. The information is a fixed string with the following format:
<pre>(1[23])</pre>
Where:
1 = a letter representing the entity ID:
P (upper case) = DomainParticipant
t = Built-in topic writer
w = built-in publication writer
r = built-in subscription writer
p (lower case) = built-in participant writer
m = peer-to-peer participant message writer
? = unknown writer
2,3 = two letters that describe the last two bits of the statusInfo inline QoS:
<table>
<tr><th>Bit 1</th><th>Bit 0</th><th>Text</th></tr>
<tr><td>0</td><td>0</td><td>__</td></tr>
<tr><td>0</td><td>1</td><td>_D</td></tr>
<tr><td>1</td><td>0</td><td>U_</td></tr>
<tr><td>1</td><td>1</td><td>UD</td></tr>
</table>
Where bit 0="Disposed" flag, and bit 1 = Unregistered flag
For example, you may see:
DATA(p[__])
DATA(p[_D])
</td></tr>
<tr><td>DATA_ BATCH</td><td>Contains information regarding the values of a batch of application data objects.</td></tr>
<tr><td>DATA_FRAG</td><td>Contains a fragment of information regarding the value of an application Data-object.
In Connext DDS 4.2e and higher, and in Connext DDS 4.5f and higher: a new format is used; captured submessages of the earlier format are displayed as DATA_FRAG_deprecated.</td></tr>
</table>

Table 5.1 RTPS 2.x Submessage Types

Submessage Type	Description
DATA_SESSION	Contains information regarding the value of an application Data-object when sent by a multi-channel Writer.
GAP	Describes the information that is no longer relevant to Readers.
HEARTBEAT	Describes the information that is available in a Writer.
HEARTBEAT_BATCH	Describes the information that is available in a Writer for batched data.
HEARTBEAT_SESSION	Describes the information that is available in a multi-channel Writer.
HEARTBEAT_VIRTUAL	Describes the information that is available from virtual Writers.
INFO_SOURCE	Provides information about the source from which subsequent Entity submessages originated.
INFO_DST	Provides information about the final destination of subsequent Entity submessages.
INFO_REPLY	Provides information about where to reply to the entities that appear in subsequent submessages. The locator provided is limited to contain a single UDPv4 address and port.
INFO_REPLY2	Provides information about where to reply to the entities that appear in subsequent submessages. The list of locators provided allows for any transport type and can accommodate 16-byte addresses.
INFO_TS[a]	Provides a source timestamp for subsequent Entity submessages.
NACK_FRAG	Provides information on the state of a Reader to a Writer.
NOKEY_DATA	Contains information regarding the value of an application Data-object that cannot be referenced by a key. In Connext DDS 4.5 and higher, as well as *RTI Data Distribution Service* 4.2e and higher, this submessage is not used.

[a]INFO_TS is an abbreviation for INFOTIMESTAMP

Table 5.1 RTPS 2.x Submessage Types

Submessage Type	Description
NOKEY_ DATA_FRAG	Contains a fragment of information regarding the value of an application data-object that cannot be referenced by a key. In Connext DDS 4.5 and higher, as well as *RTI Data Distribution Service* 4.2e and higher, this submessage is not used.
PAD	Provides padding to meet any desired memory-alignment requirements.

5.2 Displaying Packets

Wireshark has two features that make it easy to focus on packets with a particular set of values:

- **Display filters** limit the display to just packets that meet a set of criteria. See 5.2.1 Using a Display Filter.
- **Coloring rules** allow you to color-code packets based on a set of criteria so they stand out more in the full packet list. See 5.2.2 Color-Coding Packets.

For more information on filters and colors, select **Help, Wireshark Online, User's Guide** from the menubar.

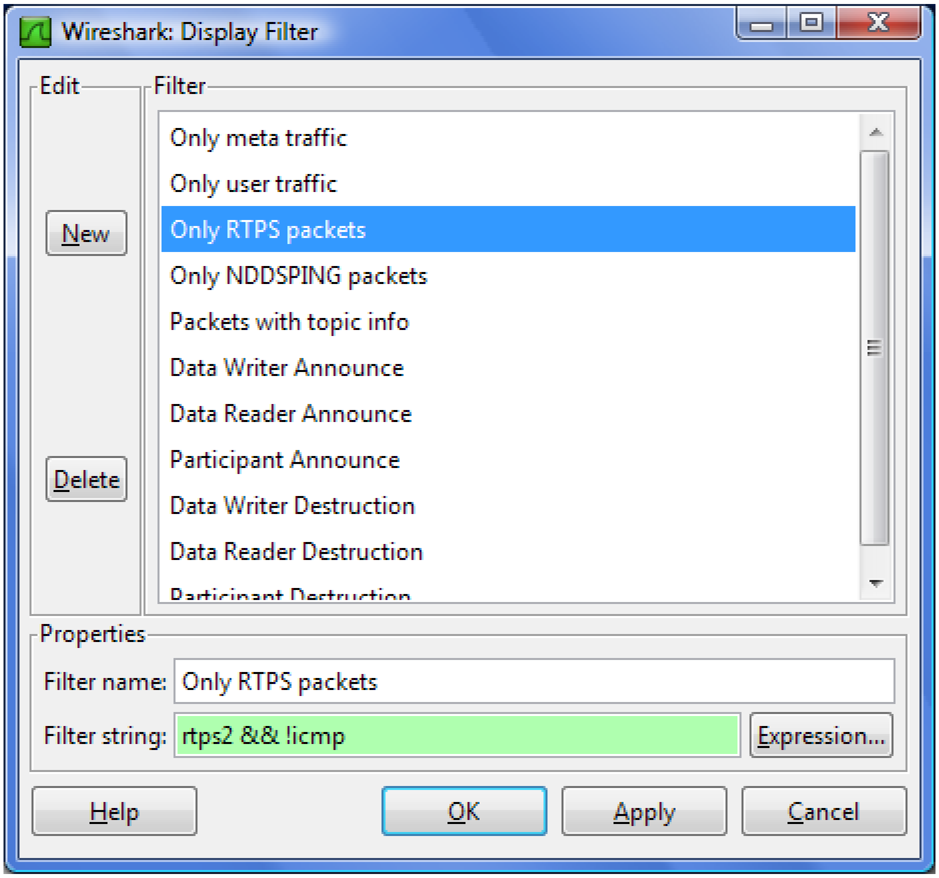
5.2.1 Using a Display Filter

A display filter only shows packets that match a certain set of criteria. You may want to start by showing only RTPS packets. *Wireshark* provides a display filter for just this purpose. There are also predefined filters for displaying just discovery (meta) traffic, or just user data traffic.

To display RTPS packets only:

1. In the main window, clear anything you have in the filter text box with the **Clear** button, then click the **Filter** button.
2. Select the preconfigured filter named "Only RTPS packets" as seen in Figure 5.1 Selecting a Display Filter.
3. Click **OK** to close the Display Filter window.

Figure 5.1 Selecting a Display Filter



As another example, let's look at how to display only RTPS packets that contain HEARTBEAT submessages.

To display HEARTBEAT packets only:

1. Clear anything you have in the filter text box with the **Clear** button, then click the **Expression...** button.
2. In the new Filter Expression window, scroll down in the **Field name** list until you see RTPS. Expand the RTPS tree (click the + sign) to see the choices for this protocol, as seen in Figure 5.2 Creating a Display Filter
3. In the **Field name** list, select **rtps.sm.id**.
4. In the **Relation** list, select ==.
5. In the **Predefined values:** list, select **HEARTBEAT**.
6. Click **OK** to close the Filter Expression window.

7. Click **Apply** in the main window to apply the new filter. Now you will see only RTPS messages that contain a HEARTBEAT submessage, as shown in Figure 5.3 Filtering by Submessage Type

Figure 5.2 Creating a Display Filter

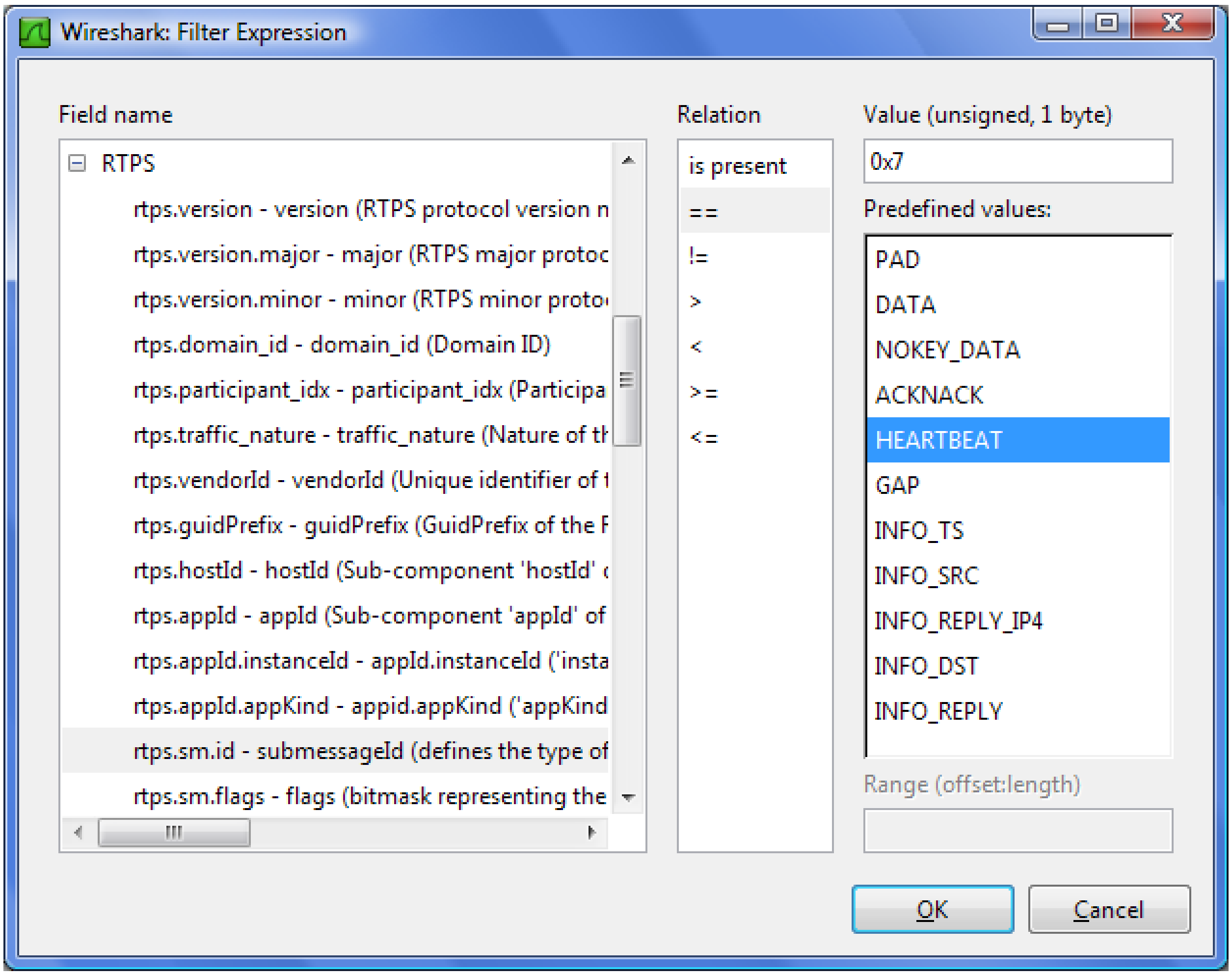
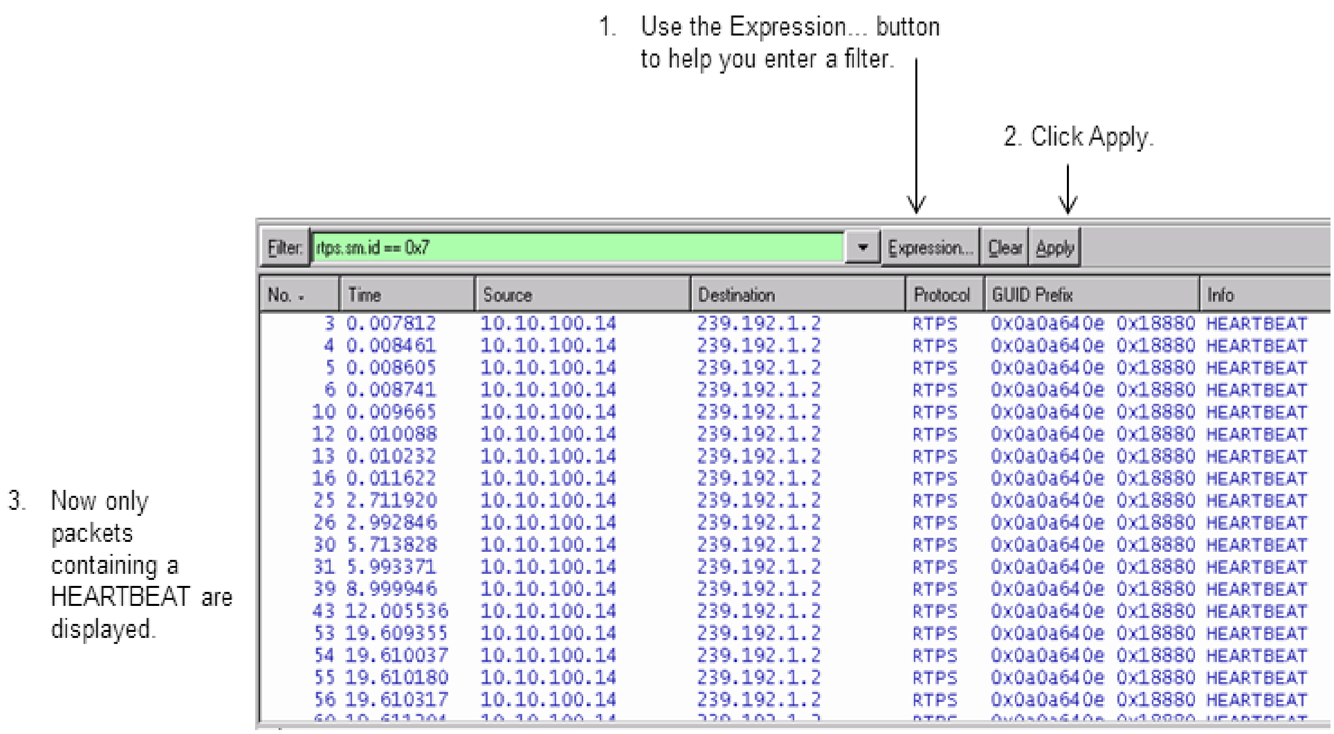


Figure 5.3 Filtering by Submessage Type



Wireshark also allows you to save filter expressions for future use. For more information, see the *Wireshark User's Guide* or online help.

Displaying RTPS Messages:

Table 5.2 , Display-Filter Fields for RTPS Messages briefly describes the meaning of each field that can be used in a display filter for RTPS. For details on the meaning of these fields, see the *Real-Time Publish-Subscribe Wire Protocol Specification* (see 1.1 Available Documentation).

Table 5.2 Display-Filter Fields for RTPS Messages

Field	Description
Header fields:	
rtps.version	Protocol version (major.minor)
rtps.version.minor	Protocol minor version
rtps.version.major	Protocol major version

Table 5.2 Display-Filter Fields for RTPS Messages

Field	Description
rtps.domain_id	Domain ID of this communication (see note below)
rtps.participant_idx	Participant index (see note below)
rtps.traffic_nature	Nature of the traffic (see note below)
rtps.vendorId	Vendor ID
rtps.guidPrefix	GUID Prefix of the packet (this does NOT match a GUID Prefix from a submessage)
rtps.hostId	Host ID component of the packet GUID Prefix
rtps.appId	App ID component of the packet GUID Prefix
rtps.appId.instanceId	Instance ID of the App Id component of the packet GUID Prefix
rtps.appId.appKind	App Kind of the App Id component of the packet GUID Prefix
Submessage-specific fields:	
rtps.sm.id	Submessage type (see Table 5.1 , RTPS 2.x Submessage Types)
rtps.sm.flags	Byte representing the submessage flags
rtps.sm.octectsToTextHeader	Value of the octetsToNextHeader from the submessage header
rtps.sm.guidPrefix	Generic GUID Prefix that appears inside a submessage (this does not match the GUID Prefix of the packet header)
rtps.sm.guidPrefix.hostId	Host ID component of the submessage GUID Prefix
rtps.sm.guidPrefix.appId	App ID component of the submessage GUID Prefix
rtps.sm.guidPrefix.appId.instanceId	InstanceId component of the App ID of the submessage GUID Prefix
rtps.sm.guidPrefix.appId.appKind	Object kind component of the App ID of the submessage GUID Prefix
rtps.sm.entityId	Object entity ID as it appear in a DATA submessage (keyHashSuffix)
rtps.sm.entityId.entityKey	'entityKey' field of the object entity ID
rtps.sm.entityId.entityKind	'entityKind' field of the object entity ID
rtps.sm.rdentityId	Reader entity ID as it appear in a submessage

Table 5.2 Display-Filter Fields for RTPS Messages

Field	Description
rtps.sm.rdentityId.entityKey	'entityKey' field of the reader entity ID
rtps.sm.rdentityId.entityKind	'entityKind' field of the reader entity ID
rtps.sm.wrentityId	Writer entity ID as it appear in a submessage
rtps.sm.wrentityId.entityKey	'entityKey' field of the writer entity ID
rtps.sm.wrentityId.entityKind	'entityKind' field of the writer entity ID
rtps.sm.seqNumber	Writer sequence number
Parameters:	
rtps.param.id	Parameter ID
rtps.param.length	Parameter length
rtps.param.ntpTime	Any generic ntpTime used in any parameter
rtps.param.ntpTime.sec	Second part of a ntpTime
rtps.param.ntpTime.fraction	Fraction part of a ntpTime
rtps.param.topicName	Topic associated with a PID_TOPIC
rtps.param.strength	Value of the strength parameter in a PID_STRENGTH
rtps.param.typeName	Value of PID_TYPE_NAME
rtps.param.userData	Raw data of PID_USER_DATA
rtps.param.groupData	Raw data of PID_GROUP_DATA
rtps.param.topicData	Raw data of PID_TOPIC_DATA
rtps.param.contentFilterName	Value of the content filter as sent in a PID_CONTENT_FILTER_PROPERTY parameter
rtps.param.relatedTopicName	Value of the related topic name as sent in a PID_CONTENT_FILTER_PROPERTY parameter
rtps.param.filterName	Value of the filter name as sent in a PID_CONTENT_FILTER_PROPERTY parameter
rtps.issueData	Value of the issue data transferred in the packets

Note: The domain_id, participant_idx, and traffic_nature are described in the latest RTPS 2 specification. The values of traffic_nature correspond to the following kinds of traffic:

- 10 = Meta Traffic Unicast
- 11 = User Traffic Unicast
- 0 = Meta Traffic Multicast
- 1 = User Traffic Multicast

The packet decoder assumes the applications are using the default value for the receive_port. Therefore, it is important to note that if the receive_port has been explicitly changed (in the locators.receive_port field of the TransportUnicast or TransportMulticast QosPolicy), then the domain_id, participant_idx, and traffic_nature values will be calculated incorrectly; in this case, these three fields should not be used in display filters nor assumed to be correct in the decoded packet view. We expect this (changing of the receive_port) to be a rare occurrence.

5.2.2 Color-Coding Packets

Wireshark allows you to display packets in different colors. Coloring rules are based on the same criteria used to create display filters (described in 5.2.1 Using a Display Filter). For instance, you can show discovery-related packets in blue and user-data packets in green. Unlike display filters, coloring rules do not hide captured packets.

Wireshark includes RTPS-related coloring rules that are automatically enabled; they are listed in Table 5.3 , Default Coloring Rules. (You can turn them off, change the colors, or edit them in other ways. See the *Wireshark User's Guide* for details.) To understand the elements in the strings, refer to the *Real-Time Publish-Subscribe Wire Protocol Specification* (see 1.1 Available Documentation). Figure 5.4 Using Coloring Rules shows a sample display.

Table 5.3 Default Coloring Rules

Coloring Rule	String
RTI TCP (purple)	rtitcp && !rtps
RTI DDSPing (green)	udp[16-23] == "nddsrtiddsping"
RTPS User traffic (red)	(rtps.sm.wrEntityId.entityKind == 0x02) || (rtps.sm.wrEntityId.entityKind == 0x03)
RTPS Meta traffic (blue)	(rtps.sm.wrEntityId.entityKind == 0xc2) || (rtps.sm.wrEntityId.entityKind == 0xc3)
Non-RTPS traffic (gray)	!rtps && !rtps2

Figure 5.4 Using Coloring Rules

24	0.140703	10.10.100.14	239.192.1.2	RTPS	0x0a0a640e	0x18880	DATA
25	2.711920	10.10.100.14	239.192.1.2	RTPS	0x0a0a640e	0x18880	HEARTBEAT
26	2.992846	10.10.100.14	239.192.1.2	RTPS	0x0a0a640e	0x18880	HEARTBEAT
27	3.022182	10.10.100.14	239.192.1.2	RTPS	0x0a0a640e	0x18880	NOKEY_DATA
28	4.022386	10.10.100.14	239.192.1.2	RTPS	0x0a0a640e	0x18880	NOKEY_DATA
29	5.022576	10.10.100.14	239.192.1.2	RTPS	0x0a0a640e	0x18880	NOKEY_DATA
30	5.713828	10.10.100.14	239.192.1.2	RTPS	0x0a0a640e	0x18880	HEARTBEAT
31	5.993371	10.10.100.14	239.192.1.2	RTPS	0x0a0a640e	0x18880	HEARTBEAT

To create a coloring rule:

1. Select **View, Coloring Rules...**, then click the **New** button to open an Edit Color Filter window.
2. Enter a name for the color filter, such as HeartBeatPackets.
3. Enter a color filter expression using the same syntax as for a display filter. If you need help, click the **Expression...** button. For examples, see Table 5.3 , Default Coloring Rules.
4. Select foreground (text) and background colors for packets that match the filter expression.

   Tip: To select a color, *click in the color-selection triangle;* use the colored circle to quickly change the contents of the triangle.
5. Click **OK** to close the Edit Color Filter window.
6. Click **Apply** in the Coloring Rules window.

   Tip: The order of the coloring rules is important. The rules are applied in the order in which they appear in the dialog box. So if there are two rules that are true for the same packet, the first will be used and the second one ignored. You can use the **Up** and **Down** buttons on the dialog to change the order of the rules.

5.3 Analyzing Packets from Connext DDS Applications

RTI's distribution of Wireshark includes two files that contain packets captured from Connext DDS 4.5f applications:

- **userDataTrace.pkt** A short trace of captured user data packets. This shows the flow of packets in an established system (after all the objects have discovered each other).
- **discoveryTrace.pkt** A longer trace of the packets sent during the discovery (startup) process.

The location of the sample files depends on your operating system:

- Linux: **/usr/share/wireshark**or **/usr/local/share/wireshark**
- OSX: **/Applications/Wireshark.app/Contents/Resources/share/wireshark/**
- Windows: **<installation directory>\RTI**

By looking at these sample files, you will learn how to:

- Load a captured sequence of packets from a file.
- Understand the flow of RTPS messages by looking at a sample sequence.
- View the contents of individual RTPS packets.

5.3.1 Analyzing the User-Data Sample Trace

Use the **File, Open...** command to open the file, **userDataTrace.pkt** (see 5.3 Analyzing Packets from Connext DDS Applications for its location).

The sample file contains a sequence of RTPS packets that illustrate the protocol when two applications use reliable communications to send/receive data.

This scenario involves two hosts, each running a Connext DDS application:

- Host 1 (10.20.1.86) is running a Connext DDS publishing application, App1.
- Host 2 (10.10.30.100) is running a Connext DDS subscribing application, App2
- The QoS for the writer and the reader have been set up to use Reliable communications.
- App1 writes user data every 4 seconds.

To create the sample capture file, Wireshark started capturing packets on the subscribing host after the discovery process completed, using the following capture filter:

```
rtps && (ip.src == 10.10.30.100 || ip.dst == 10.10.30.100)
```

Figure 5.5 User Data Sample Packets shows the packets captured by *Wireshark*, which includes three types of RTPS packets:

- Data from the writer to the reader
- Acknowledgements from the reader to the writer
- Heartbeats sent regularly from the writer to the reader

Figure 5.5 User Data Sample Packets

No.	Time	Source	Destination	Protocol	GUID prefix	Info
1	0.000000	10.20.1.86	10.10.30.100	RTPS2		INFO_TS, DATA
2	2.000907	10.20.1.86	10.10.30.100	RTPS2		INFO_DST, HEARTBEAT
3	2.001181	10.10.30.100	10.20.1.86	RTPS2		INFO_DST, ACKNACK
4	4.000938	10.20.1.86	10.10.30.100	RTPS2		INFO_TS, DATA
5	5.003878	10.20.1.86	10.10.30.100	RTPS2		INFO_DST, HEARTBEAT
6	5.004171	10.10.30.100	10.20.1.86	RTPS2		INFO_DST, ACKNACK
7	8.001939	10.20.1.86	10.10.30.100	RTPS2		INFO_TS, DATA
8	8.005850	10.20.1.86	10.10.30.100	RTPS2		INFO_DST, HEARTBEAT
9	8.006104	10.10.30.100	10.20.1.86	RTPS2		INFO_DST, ACKNACK

Table 5.4 , Analysis of User Data Sample Trace and Figure 5.6 User Data Sample Packet Flow describe the trace shown in Figure 5.5 User Data Sample Packets.

Table 5.4 Analysis of User Data Sample Trace

Direction	Packet #	Description
App1 → App2	1	Data packet sent to the reader, containing both INFO_TS and DATA submessages. Packet has sequence number = 2 (expand the protocol tree in the Packet Details pane and check the writerSeqNumber value, as seen in Figure 5.7 Examining Packet Details).
	2	HEARTBEAT from writer to reader.
App1 ← App2	3	ACKNACK to acknowledge all data packets up to, but not including, sequence number 3 (expand the protocol tree in the Packet Details pane and check the readerSNState.bitmapBase value).
App1 → App2	4	Another data packet (sequence number 3).
	5	HEARTBEAT from writer to reader.
App1 ← App2	6	ACKNACK to acknowledge packet #4.
App1 → App2	7	Another data packet (sequence number 4).
	8	HEARTBEAT from writer to reader.
App1 ← App2	9	ACKNACK to acknowledge packet #7.

Figure 5.6 User Data Sample Packet Flow

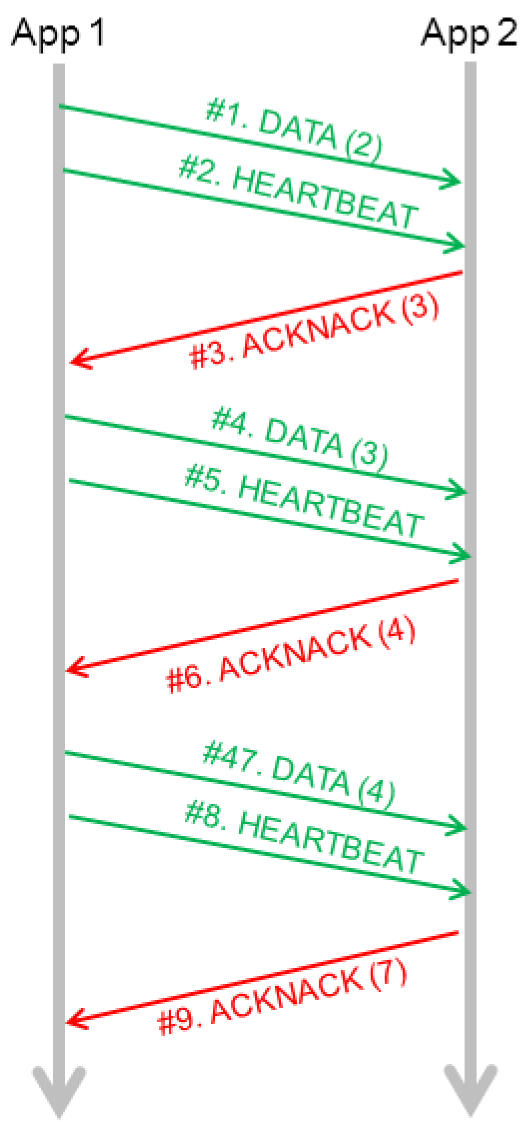
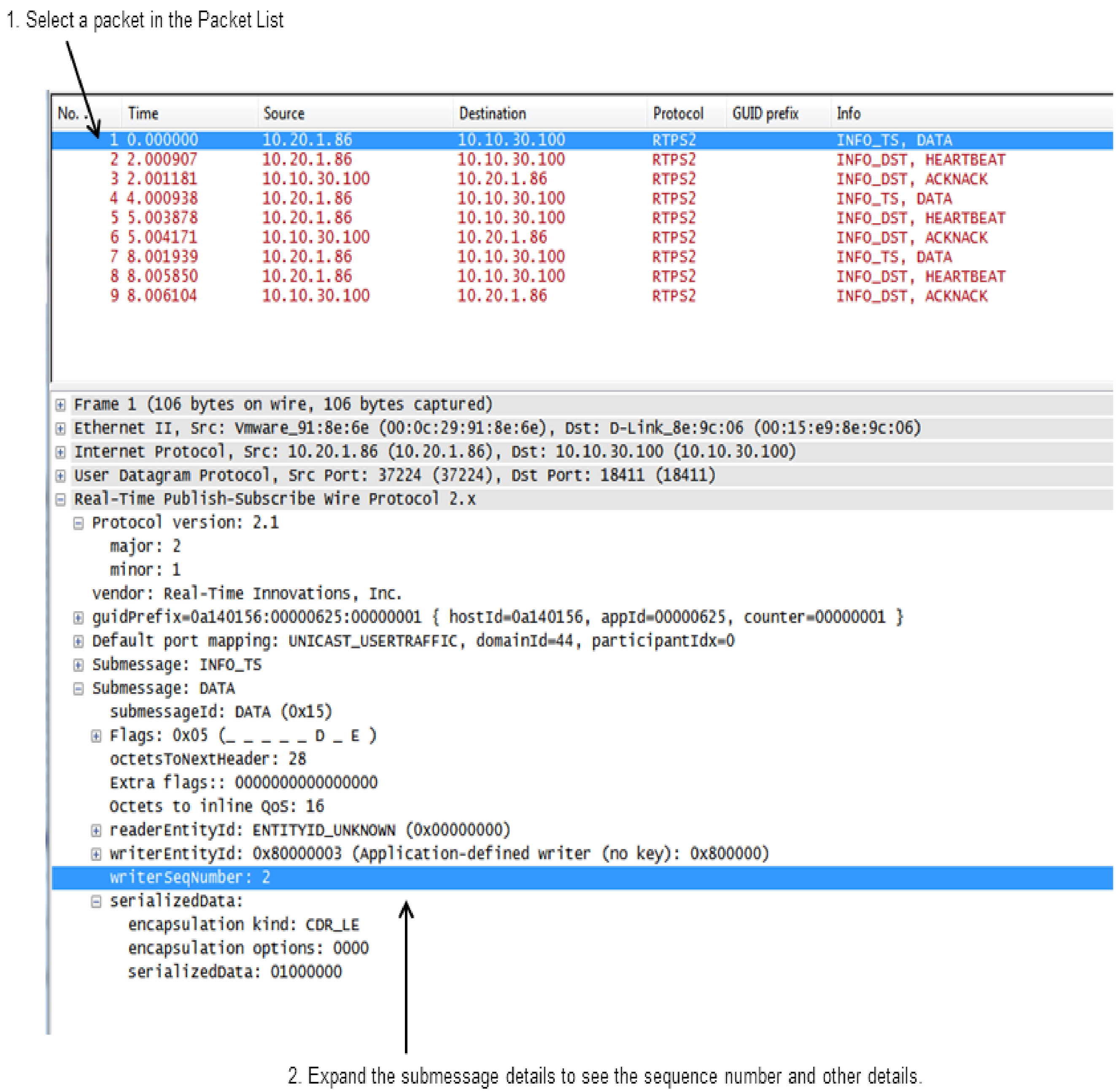


Figure 5.7 Examining Packet Details



5.3.2 Analyzing the Discovery-Data Sample Trace

Use the **File, Open...** command to open the file, **discoveryTrace.pkt** (see 5.3 Analyzing Packets from Connext DDS Applications for its location).

The sample file contains a sequence of RTPS packets that illustrate the protocol when two applications use best-effort communications to send/receive data.

This scenario involves two hosts, each running one Connext DDS application.

- Host 1 (10.10.100.65) is running a Connext DDS publishing application, App1.
- Host 2 (10.10.30.100) is running a Connext DDS subscribing application, App2.
- Both applications have a maximum participant index of 1 and have each other in their initial_peer_list.
- All but one QoS are at default values; this default includes the use of automatic discovery via the default UDPv4 transport. The one non-default QoS is having both applications' initial_participant_announcements set to 1, as this reduces the number of redundant announcements in the example trace.

Wireshark was set up to start capturing packets *before* either application was started. The publishing application was started first, followed (about 6 seconds later) by the subscribing application. Figure 5.8 Discovery Data Sample File shows the packets captured by Wireshark.

Let's walk through the RTPS packets to understand what occurred in this sequence. Table 5.5 , Analysis of Sample File's Packets describes what happened (non-RTPS packets are omitted). In the table, the term "meta DATA" refers to DATA packets containing meta (discovery) data (as opposed to user data).

Table 5.5 Analysis of Sample File's Packets

Direction	Packet #	Description
App1 → App2	1 - 12	When the writer participant starts, Connext DDS announces the creation of a new participant to all potential participants in the initial_peer_list. Potential participants are initially calculated as: for each peer in initial_peer_list, peer/participant(i), where i <= maximum participant index. Since the participant's maximum participant index is 1 and the initial_peer_list contains only 10.10.30.100, the potential participant list is {10.10.30.100/participant(0), 10.10.30.100/participant(1)}. Since each participant gets its own receive locator, we send separate (but identical) packets to each potential participant listening on its own locator. Before sending packets with participant information, the participant sends to each receive locator a PING packet (as packets #1 and #2). These packets serve to prime ARP tables and to see if the locators are reachable destinations. Since there is no other Connext DDS application in the system in the same domain, these locators are unreachable (as packets #3 and #4). Because the participant is newly created, it sends its information to each locator (as packets #5 and #6). These are to unreachable destinations (packets #7 and #8). It then again sends its information (packets #9 and #10), in accordance to its initial_participant_announcement QoS of 1. Again these are to unreachable destinations (packets #11 and #12).
App1 ← App2	13-18	Similar to the writer participant, when the reader participant starts, Connext DDS announces the new participant. Note: the destination of the writer participant is reachable, so no "unreachable destination" packets are generated.

Table 5.5 Analysis of Sample File's Packets

Direction	Packet #	Description
App1 → App2	19-20	When the writer participant learns about the new reader participant, it uses HEARTBEATs and ACKNACKs from its builtin discovery endpoints to tell the reader participant about the writer and readers it has. First, it tells the reader participant about its builtin participant liveliness reader (packet #19) and writer (packet #20).
	21-22	Repeat announcements about the writer participant, which reduce the chance that the newly created reader participant will drop the reply from the writer participant.
	23-24	HEARTBEATs telling the reader participant how many readers (0) and writers (1) it has. The reader participant will know from this that it has to get a meta DATA from the writer. Note: to be precise, each meta DATA reflects a change to the state of the writer or reader. The number of writer samples would increase if the writer was changed or deleted.
	25	Writer participant packet #22 for reader participant index 1 was not reachable, because the reader participant was created with participant id 0.
	26-27	ACKNACKs telling the reader participant that builtin discovery readers for publications and subscriptions are created.
App1 ← App2	28	Reader participant begins to announce its builtin discovery endpoints after having received the writer participant's announcements. ACKNACK from the reader participant's builtin participant liveliness.
	29-30	Re-announcements of reader participant info, triggered by receiving the writer announcements.
	31-35	Initial HEARTBEATs and ACKNACKs from the reader participant's builtin discovery endpoints, indicating it has one reader and no writers.
	36	ACKNACK in response to HEARTBEAT packet #24, requesting that the writer participant's builtin publication writer resend the meta DATA for its writer
	37-38	HEARTBEATs sent in response to writer ACKNACKs, announcing that the reader participant has one reader and no writers.

Table 5.5 Analysis of Sample File's Packets

Direction	Packet #	Description
App1 → App2	39	Initial HEARTBEAT from participant liveliness writer, of no samples written.
App1 ← App2	40	ACKNACK from participant liveliness reader, of no samples received.
App1 → App2	41	ACKNACK from writer participant for reader participant to resend the meta DATA for its reader
	42-43	Further HEARTBEATs from the reader participant
App1 ← App2	44	Resent meta DATA about the reader, sent in response to the ACKNACK of packet 41
App1 → App2	45	A resent meta DATA about the writer, sent in response to the ACKNACK of packet 36
App1 ↔App2	46-49	ACKNACKs that all resent DATA was received successfully
App1 → App2	50	PING from the application writer to the application reader's locator
	51-53	Writer finally sends user DATA to the reader

Figure 5.8 Discovery Data Sample File

No.	Time	Source	Destination	Protocol	GUID prefix	Info
1	0.000000	10.10.100.65	10.10.30.100	RTPS2		PING
2	0.000045	10.10.100.65	10.10.30.100	RTPS2		PING
3	0.000130	10.10.30.100	10.10.100.65	ICMP		Destination unreachable (Port unreachable)
4	0.000161	10.10.30.100	10.10.100.65	ICMP		Destination unreachable (Port unreachable)
5	0.008788	10.10.100.65	10.10.30.100	RTPS2		INFO_TS, DATA(p)
6	0.008801	10.10.100.65	10.10.30.100	RTPS2		INFO_TS, DATA(p)
7	0.009079	10.10.30.100	10.10.100.65	ICMP		Destination unreachable (Port unreachable)
8	0.009098	10.10.30.100	10.10.100.65	ICMP		Destination unreachable (Port unreachable)
9	0.017908	10.10.100.65	10.10.30.100	RTPS2		INFO_TS, DATA(p)
10	0.017923	10.10.100.65	10.10.30.100	RTPS2		INFO_TS, DATA(p)
11	0.018227	10.10.30.100	10.10.100.65	ICMP		Destination unreachable (Port unreachable)
12	0.018254	10.10.30.100	10.10.100.65	ICMP		Destination unreachable (Port unreachable)
13	5.423355	10.10.30.100	10.10.100.65	RTPS2		PING
14	5.423415	10.10.30.100	10.10.100.65	RTPS2		PING
15	5.423977	10.10.30.100	10.10.100.65	RTPS2		INFO_TS, DATA(p)
16	5.424000	10.10.30.100	10.10.100.65	RTPS2		INFO_TS, DATA(p)
17	5.427251	10.10.30.100	10.10.100.65	RTPS2		INFO_TS, DATA(p)
18	5.427271	10.10.30.100	10.10.100.65	RTPS2		INFO_TS, DATA(p)
19	5.441956	10.10.100.65	10.10.30.100	RTPS2		INFO_DST, ACKNACK
20	5.446546	10.10.100.65	10.10.30.100	RTPS2		INFO_DST, HEARTBEAT
21	5.446635	10.10.100.65	10.10.30.100	RTPS2		INFO_TS, DATA(p)
22	5.446664	10.10.100.65	10.10.30.100	RTPS2		INFO_TS, DATA(p)
23	5.446779	10.10.100.65	10.10.30.100	RTPS2		INFO_DST, HEARTBEAT
24	5.446911	10.10.100.65	10.10.30.100	RTPS2		INFO_DST, HEARTBEAT
25	5.446963	10.10.30.100	10.10.100.65	ICMP		Destination unreachable (Port unreachable)
26	5.447070	10.10.100.65	10.10.30.100	RTPS2		INFO_DST, ACKNACK
27	5.447220	10.10.100.65	10.10.30.100	RTPS2		INFO_DST, ACKNACK
28	5.447630	10.10.30.100	10.10.100.65	RTPS2		INFO_DST, ACKNACK
29	5.447871	10.10.30.100	10.10.100.65	RTPS2		INFO_TS, DATA(p)
30	5.447886	10.10.30.100	10.10.100.65	RTPS2		INFO_TS, DATA(p)
31	5.447909	10.10.30.100	10.10.100.65	RTPS2		INFO_DST, HEARTBEAT
32	5.448129	10.10.30.100	10.10.100.65	RTPS2		INFO_DST, HEARTBEAT
33	5.448144	10.10.30.100	10.10.100.65	RTPS2		INFO_DST, HEARTBEAT
34	5.448154	10.10.30.100	10.10.100.65	RTPS2		INFO_DST, ACKNACK
35	5.448374	10.10.30.100	10.10.100.65	RTPS2		INFO_DST, ACKNACK
36	5.448588	10.10.30.100	10.10.100.65	RTPS2		INFO_DST, ACKNACK
37	5.448606	10.10.30.100	10.10.100.65	RTPS2		INFO_DST, HEARTBEAT
38	5.448617	10.10.30.100	10.10.100.65	RTPS2		INFO_DST, HEARTBEAT
39	5.449305	10.10.100.65	10.10.30.100	RTPS2		INFO_DST, HEARTBEAT
40	5.449490	10.10.30.100	10.10.100.65	RTPS2		INFO_DST, ACKNACK
41	5.457424	10.10.100.65	10.10.30.100	RTPS2		INFO_DST, ACKNACK
42	5.457532	10.10.100.65	10.10.30.100	RTPS2		INFO_DST, HEARTBEAT
43	5.457603	10.10.100.65	10.10.30.100	RTPS2		INFO_DST, HEARTBEAT
44	5.457623	10.10.30.100	10.10.100.65	RTPS2		INFO_TS, INFO_DST, DATA(r), HEARTBEAT
45	5.457695	10.10.100.65	10.10.30.100	RTPS2		INFO_TS, INFO_DST, DATA(w), HEARTBEAT
46	5.457796	10.10.100.65	10.10.30.100	RTPS2		INFO_DST, ACKNACK
47	5.457948	10.10.30.100	10.10.100.65	RTPS2		INFO_DST, ACKNACK
48	5.458131	10.10.100.65	10.10.30.100	RTPS2		INFO_DST, ACKNACK
49	5.458150	10.10.30.100	10.10.100.65	RTPS2		INFO_DST, ACKNACK
50	5.460151	10.10.100.65	10.10.30.100	RTPS2		PING
51	8.037246	10.10.100.65	10.10.30.100	RTPS2		INFO_TS, DATA
52	12.037962	10.10.100.65	10.10.30.100	RTPS2		INFO_TS, DATA
53	16.037869	10.10.100.65	10.10.30.100	RTPS2		INFO_TS, DATA

Chapter 6 Practical Uses with RTI Applications

This section offers a few suggestions on how you can use *Wireshark* during Connext DDS application development:

6.1 Debugging Discovery Problems

While many object discovery problems are difficult to diagnose, others are quite obvious once you use the right diagnostic tools. By inspecting all RTPS packets with *Wireshark*, you may be able to narrow the problem down to one of the following:

The participants are not discovering each other. In this case, you will see periodic sending of DATA packets, but no response from the other host that is not being discovered.

The participants have discovered each other, but their contained readers/writers are not getting hooked up correctly. In this case, you may see HEARTBEAT and ACKNACK packets for the reserved meta-data representing the reader and writer from one participant to another, but the other participant is not responding back in accordance to the RTPS protocol.

The objects have all discovered each other, but the writer is not sending user-data. In this case, you will see the discovery protocol complete successfully, but not see DATA packets containing user data from the writer.

When a participant containing a writer sends meta data to other participants, and those other participants respond with ACKNACK packets to acknowledge those discovery packets, all you can say is that the declaration for that writer was received by all participants in the system. But just because a participant is writing DATA packets does not necessarily mean it is writing your application's user data. Connext DDS also uses DATA packets to propagate internal object information. When in doubt, check the *traffic_nature* field in the decoded packet to see how the packet is being used.

A subscriber reciprocally declares its reader object with another DATA packet to all concerned participants. This happens before the writer application starts publishing user data. Connext DDS uses separate built-in objects to announce and discover readers vs. writers, so it's important to check the *writerEntityId* of the DATA packet to confirm that the participants in question have discovered the reader/writer correctly.

Lastly, it's important to check whether the topic and type declared in the meta data of the reader matches that in the meta data of the writer. Assuming that neither party is deliberately ignoring certain Entities (e.g. Participant, Topic, DataReader, DataWriter), if all these were acknowledged (with ACKNACK packets), the reader participant should at this point be ready to accept user data from the writer, and the writer will send the data to the reader. Exactly when the data will appear on the wire will depend on when the writer writes the next sample, as well as the QoS of both the reader and writer.

The writer is writing your data, but the reader is not able to access that data when it calls read() or take(). In this case, you should check your QoS settings. Compare the writer's QoS against the reader's. Perhaps the *minimum_separation* in the TimeBasedFilter QosPolicy of the reader is inadvertently filtering out received issues.

Once a writer is writing user data to a data reader, the initial discovery phase is over. But there can be an "anti-discovery" problem: depending on the Liveliness QoS, Connext DDS may purge a remote entity that it considers to be stale. Regardless of what kind of liveliness setting you use, the main idea is to ensure that your participant and its entities renew their liveliness (automatically or manually) within the declared duration. A classic symptom of communication ceasing due to a liveliness expiration is that a participant stops sending its periodic participant DATA packet. (See the *RTI Core Libraries and Utilities User's Manual* or online documentation for information about the Liveliness QosPolicy.)

NOTE: Connext DDS can log more detailed information about what it is doing at higher verbosity settings. See the *RTI Connext DDS Core Libraries User's Manual*'s Troubleshooting chapter for more information on setting verbosity.

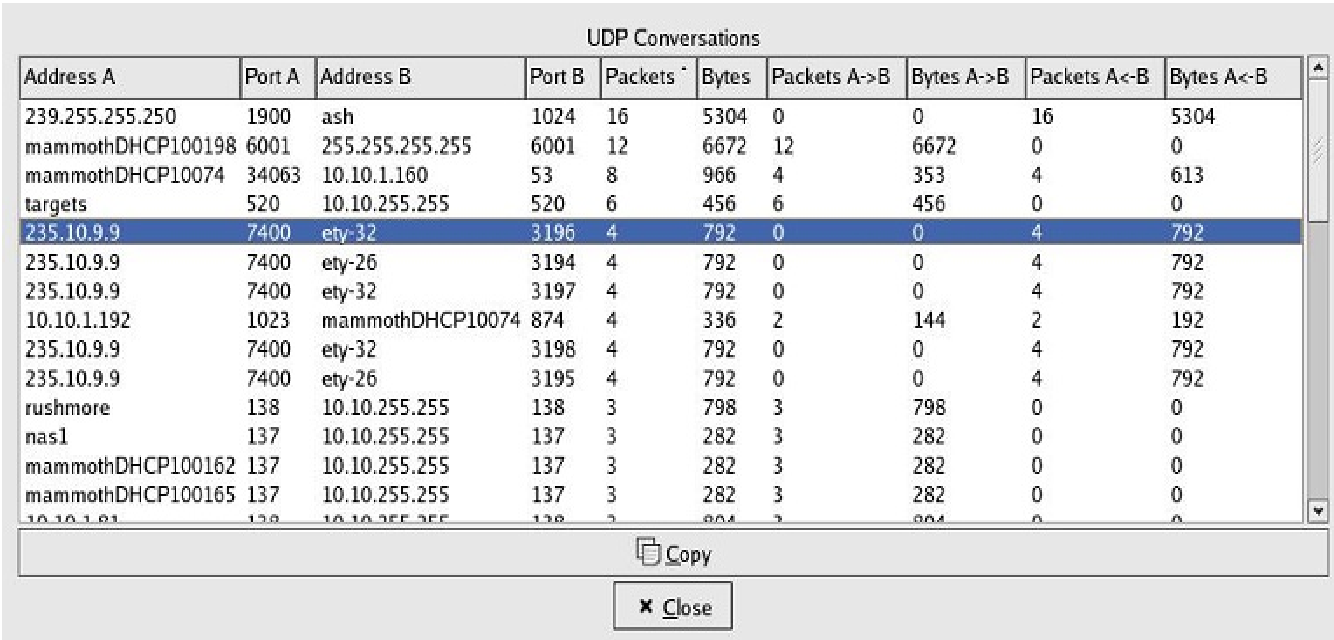
6.2 Visualizing Your System

Once your applications are communicating, tuning Connext DDS to maximize performance may require an in-depth understanding of your network. A visual understanding of Connext DDS network usage is very valuable for system tuning.

For example, you may be sending data as fast as Connext DDS will allow and wonder why the reader cannot keep up. *Wireshark* itself offers many statistical analysis tools under the Statistics menu.

As Figure 6.1 UDP Conversations through Figure 6.3 Protocol Hierarchy show, you can see how many RTPS packets are being sent, what portion of total network bandwidth RTPS packets are taking up, which hosts are talking to others, and how much bandwidth is being used to do so. In our "sending too fast" example, you may find that the RTPS packets are being dropped at a host with a relatively slow reader. In some extreme cases, even *Wireshark* may not see all the packets sent, because the network card on the sniffing machine itself dropped them.

Figure 6.1 UDP Conversations



UDP Conversations

Address A	Port A	Address B	Port B	Packets	Bytes	Packets A->B	Bytes A->B	Packets A<-B	Bytes A<-B
239.255.255.250	1900	ash	1024	16	5304	0	0	16	5304
mammothDHCP100198	6001	255.255.255.255	6001	12	6672	12	6672	0	0
mammothDHCP10074	34063	10.10.1.160	53	8	966	4	353	4	613
targets	520	10.10.255.255	520	6	456	6	456	0	0
235.10.9.9	7400	ety-32	3196	4	792	0	0	4	792
235.10.9.9	7400	ety-26	3194	4	792	0	0	4	792
235.10.9.9	7400	ety-32	3197	4	792	0	0	4	792
10.10.1.192	1023	mammothDHCP10074	874	4	336	2	144	2	192
235.10.9.9	7400	ety-32	3198	4	792	0	0	4	792
235.10.9.9	7400	ety-26	3195	4	792	0	0	4	792
rushmore	138	10.10.255.255	138	3	798	3	798	0	0
nas1	137	10.10.255.255	137	3	282	3	282	0	0
mammothDHCP100162	137	10.10.255.255	137	3	282	3	282	0	0
mammothDHCP100165	137	10.10.255.255	137	3	282	3	282	0	0

Copy

Close

Figure 6.2 I/O Graph

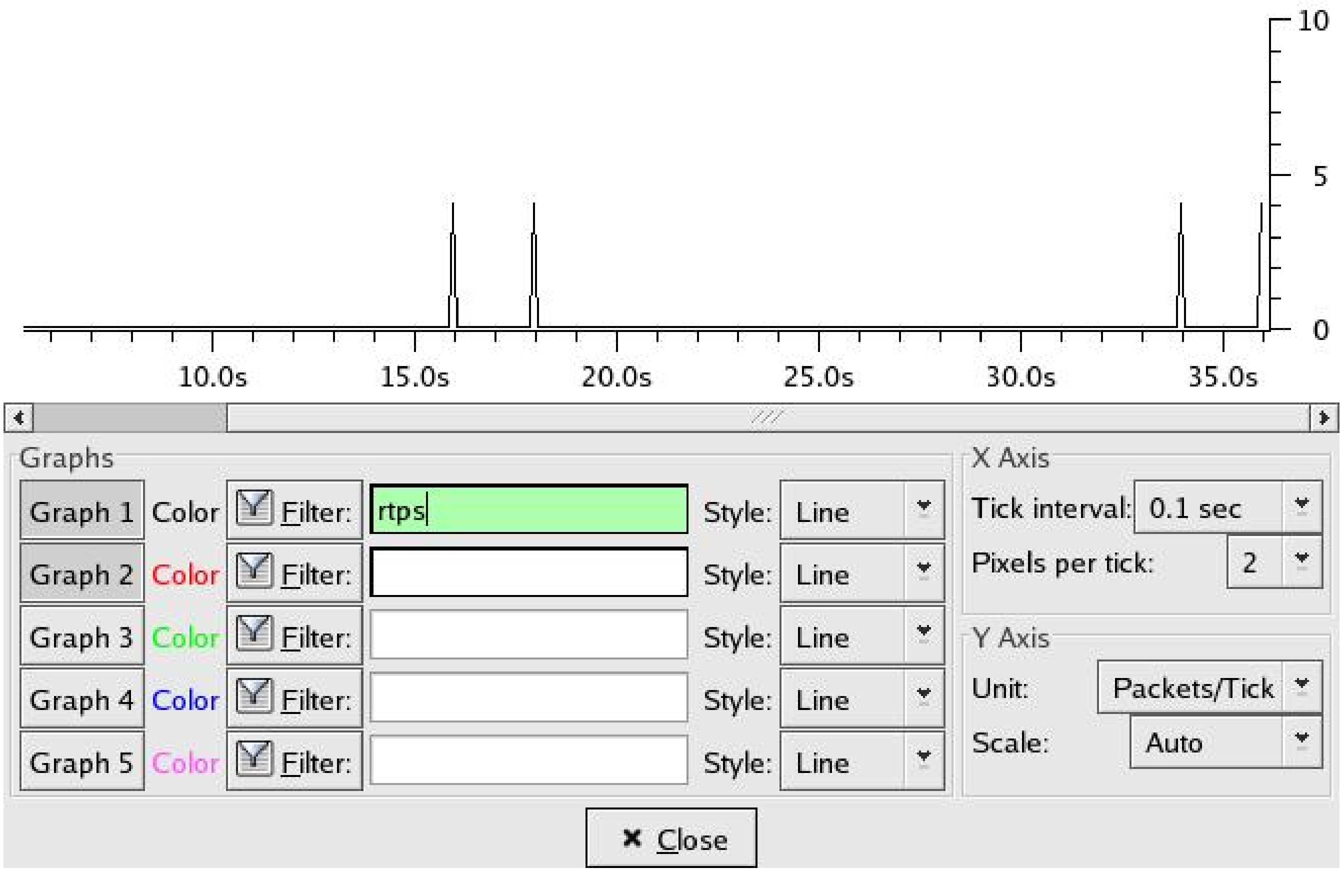


Figure 6.3 Protocol Hierarchy

Protocol	% Packets	Packets	Bytes	Mbit/s	End Packets	End Bytes	End Mbit/s
▽ Frame	100.00%	128	25957	0.005	0	0	0.000
▽ Linux cooked-mode capture	100.00%	128	25957	0.005	0	0	0.000
▽ Internet Protocol	100.00%	128	25957	0.005	0	0	0.000
▽ User Datagram Protocol	100.00%	128	25957	0.005	0	0	0.000
Data	20.31%	26	7636	0.001	26	7636	0.001
Real-Time Publish-Subscribe Wire Protocol	15.62%	20	3960	0.001	20	3960	0.001
Routing Information Protocol	4.69%	6	456	0.000	6	456	0.000
▽ Remote Procedure Call	21.88%	28	3252	0.001	0	0	0.000
Yellow Pages Service	21.88%	28	3252	0.001	28	3252	0.001
Domain Name Service	6.25%	8	966	0.000	8	966	0.000
▽ NetBIOS Datagram Service	10.16%	13	3353	0.001	0	0	0.000
▽ SMB (Server Message Block Protocol)	10.16%	13	3353	0.001	0	0	0.000
▽ SMB MailSlot Protocol	10.16%	13	3353	0.001	0	0	0.000
Microsoft Windows Browser Protocol	10.16%	13	3353	0.001	13	3353	0.001
NetBIOS Name Service	7.03%	9	846	0.000	9	846	0.000
Hypertext Transfer Protocol	12.50%	16	5304	0.001	16	5304	0.001
Network Time Protocol	1.56%	2	184	0.000	2	184	0.000

OK

6.3 Providing Information to RTI Support

If you ever need to contact RTI Support for an issue related to Connext DDS, the captured packets will help RTI support diagnose the problem faster (especially when accompanied by a Connext DDS log created with a high verbosity setting).

See the *RTI Connext DDS Core Libraries User's Manual's* Troubleshooting section for more information on setting verbosity.