

*Using Wireshark™
with
RTI Data Distribution Service*

Getting Started Guide

Version 1.2





© 2005-2010 Real-Time Innovations, Inc.
All rights reserved.
Printed in U.S.A. First printing.
June 2010.

Trademarks

Real-Time Innovations and RTI are registered trademarks of [Real-Time Innovations, Inc.](#)
All other trademarks used in this document are the property of their respective owners.

Copy and Use Restrictions

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form (including electronic, mechanical, photocopy, and facsimile) without the prior written permission of Real-Time Innovations, Inc.

RTI Data Distribution Service software is furnished under and subject to the RTI software license agreement. The software may be used or copied only under the terms of the license agreement.

Wireshark is Open Source software released under the terms of the GNU General Public License (version 2) as published by the [Free Software Foundation](#).

Technical Support

Real-Time Innovations, Inc.

385 Moffett Park Drive

Sunnyvale, CA 94089

Phone: (408) 990-7444

Email: support@rti.com

Website: <http://www.rti.com/support>

Contents

1	Introduction	1-1
1.1	Available Documentation.....	1-3
1.2	Reading Guide	1-3
1.3	How to Get Support.....	1-4
2	Installation	2-1
2.1	Before Installation.....	2-1
2.2	Installing Wireshark on Windows Systems	2-1
2.3	Installing Wireshark on Linux Systems.....	2-2
2.4	Installing Wireshark on Solaris Systems	2-2
2.5	Uninstalling Wireshark.....	2-2
3	Starting Wireshark	3-1
4	Capturing RTPS Packets	4-1
5	Analyzing RTPS Packets	5-1
5.1	RTPS Submessage Types	5-2
5.2	Displaying Packets	5-2
5.2.1	Using a Display Filter.....	5-4
5.2.2	Color-Coding Packets	5-10
5.3	Analyzing Packets from RTI Data Distribution Service Applications.....	5-11
5.3.1	Analyzing the User Data Sample Trace	5-12
5.3.2	Analyzing the Discovery Data Sample Trace	5-16

6	Practical Uses with RTI Applications.....	6-1
6.1	Debugging Discovery Problems.....	6-1
6.2	Visualizing Your System.....	6-3
6.3	Providing Information to RTI Support.....	6-5

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® Data Distribution Service* middleware.

Wireshark can be used to capture and analyze RTPS packets from *RTI Data Distribution Service* 4.x 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.
- ❑ A column in the Packet List that shows the GUID Prefix for each RTPS packet. (This value uniquely identifies a DomainParticipant within a Domain.)
- ❑ Coloring rules that highlight important RTPS packets. Packets not strictly related to *RTI Data Distribution Service* traffic are grayed-out.

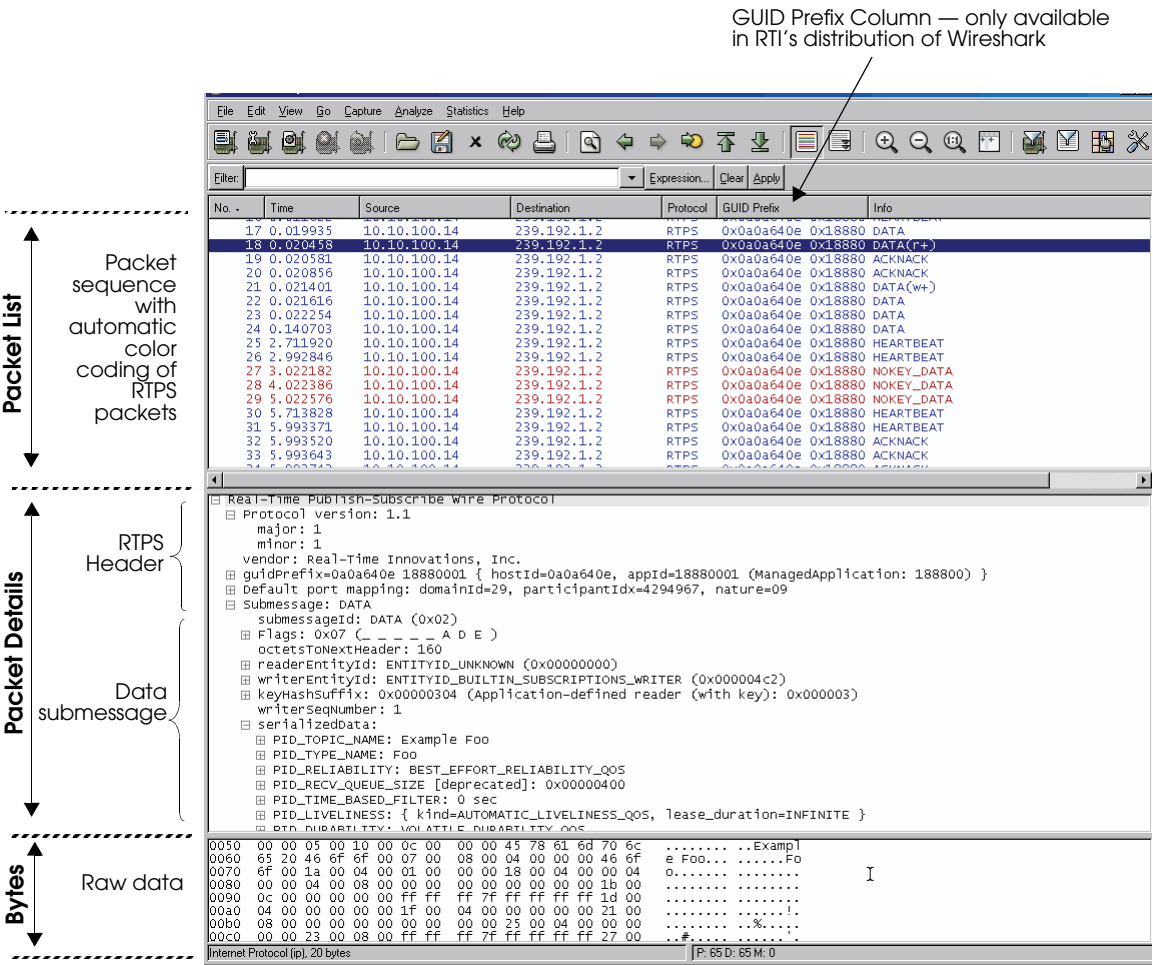
Network packet and traffic analyzers are used during application development and distributed system configuration to monitor the packets over the network. You can use filters to capture specific types of packets, then analyze the captured packets.

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 *RTI Data Distribution Service* applications.
- ❑ A detailed look at the contents of individual packets.

Wireshark's main window makes it easy to see both views. [Figure 1.1](#) shows the main window and points out some important features.

Figure 1.1 Wireshark's Main Window



The Packet List pane on top lists the sequence of captured submessages. The middle pane shows a decoded view of a selected packet. The bottom pane shows the raw data for any selected field in a packet. You can customize these views from the View menu. To change which columns are displayed, select Edit, Preferences, User Interface, Columns.

1.1 Available Documentation

- ❑ *RTI Wireshark Release Notes*. Please see **RTI_Wireshark_ReleaseNotes.pdf** for system requirements, installation instructions, and other important information.
- ❑ *RTI Data Distribution Service User's Manual*. This document provides details on the *RTI Data Distribution Service* API and describes how RTPS packets are used by *RTI Data Distribution Service*-based applications. In particular, you should review the *Discovery* chapter. Open **<NDDSHOME>/doc/pdf/RTI_DDS_UsersManual.pdf**, where **<NDDSHOME>** is where you installed *RTI Data Distribution Service*.
- ❑ *RTPS Specification*. Please see <http://www.omg.org/spec/DDS/2.1/>.
- ❑ *Wireshark online help*. There is extensive online help included with Wireshark. Select **Help**, **Contents** from the menubar for a detailed user's guide in HTML format.
- ❑ *Wireshark User's Guide*. This PDF document 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 chapter 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 *RTI Data Distribution Service* 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 Solaris systems, or as administrator on Windows systems.

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

2.1 Before Installation

If you have *Ethereal*, *Wireshark*, or *RTI Protocol Analyzer with Wireshark* installed, we highly recommend that you remove them before installing RTI's distribution of *Wireshark*.

On Solaris systems: If an older version of *Wireshark* is already installed, uninstall it by running this command:

```
pkgrm wireshark
```

2.2 Installing Wireshark on Windows Systems

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

Install *Wireshark* using the RedHat Package Manager (RPM):

1. Login as super-user.
2. `cd <location of the distribution file>`
3. `rpm -i Wireshark-<version>-<architecture>.rpm`

For more information on installing RPMs, please see <http://www.rpm.org>.

2.4 Installing Wireshark on Solaris Systems

Before Installation:

- ☐ Make sure you have installed the required packages listed in [Section 1.1 in the Release Notes](#).
- ☐ Make sure you have modified the font cache configuration file and rebuilt the font cache (see [Section 1.1.1 in the Release Notes](#)).
- ☐ Make sure you have *root* privileges.

1. `cd <location of the distribution file>`
 2. `gunzip Wireshark-<version>-<architecture>.gz`
 3. `pkgadd -d Wireshark-<version>-<architecture>`
-

2.5 Uninstalling Wireshark

To uninstall Wireshark:

- ☐ **On Windows systems:** From the **Start** menu, select **Control Panel, Add/Remove Programs** (or **Programs and Features**), **Wireshark**.
- ☐ **On Linux and Solaris systems:** While logged in as *root*, enter:
`rpm -e wireshark`

Chapter 3 Starting Wireshark

Important!

To capture packets from the network, you must run Wireshark as root/administrator.

On Linux systems:

```
# /usr/bin/wireshark &
```

On Solaris systems:

```
# /usr/local/bin/wireshark &
```

On Windows systems:

Use the **Start** menu to select **Wireshark**.

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](#) 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 an RTI Data Distribution Service application:

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

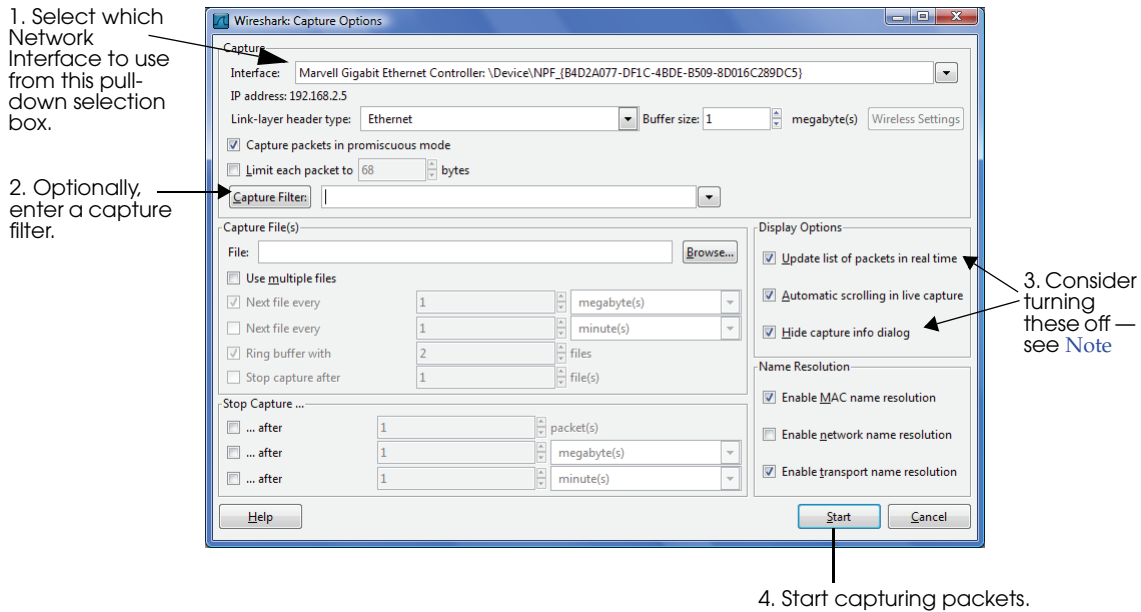
The defaults in the Capture Option window may very well suit your needs—they will capture all packets sent to the selected interface. Then you can filter the displayed results with a display filter, as described in [Section 5.2.1](#).

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


4. Click **Start** to start the capture session.

Note The "Update list of packets in real-time" and "Hide capture info dialog" check boxes 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.

Figure 4.1 Starting a 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 steps above, 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](#)) before you press **Capture**. [Table 4.1](#) provides some examples.

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

Table 4.1 Example Capture Filters

To Capture ...	Enter ...
Capture only RTPS	udp[8:4] == 0x52545053 or (ip[6:2] & 0x1FFF != 0)
Only UDP packets	udp
Only UDP multicast packets	udp and ip multicast
Only non-UDP multicast	udp and not ip multicast
Only UDP from/to 10.10.1.192	udp and host 10.10.1.192
Only packets from IP addresses 10.10.100.14 to 10.10.100.74	src host 10.10.100.14 and dst host 10.10.100.74

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 **wireshark** with the **-f** argument to try a capture filter expression. (See **Help, Manual Pages, wireshark**.)

For help analyzing captured RTPS packets, see [Chapter 5](#).

Chapter 5 Analyzing RTPS Packets

This chapter 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 chapter 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 Object Discovery chapter in the *RTI Data Distribution Service User’s Manual*.

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

This chapter 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 [Available Documentation \(Section 1.1\)](#)).

This chapter includes the following sections:

- ❑ [RTPS Submessage Types \(Section 5.1\)](#)
- ❑ [Displaying Packets \(Section 5.2\)](#)
- ❑ [Analyzing Packets from RTI Data Distribution Service Applications \(Section 5.3\)](#)

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 [Figure 5.1](#)) lists the types of submessages in each packet.

Figure 5.1 Analyzing Packets

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	5.712828	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](#) 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*.

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 [Section 5.2.1](#).
- ❑ **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 [Section 5.2.2](#).

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

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																		
DATA	<p>Contains information regarding the value of an application Data-object. The information is a fixed string with the following format:</p> <p style="padding-left: 40px;">(1 [2 3])</p> <p>Where:</p> <p>1 = a letter representing the entity ID:</p> <p style="padding-left: 40px;">P (upper case) = DomainParticipant</p> <p style="padding-left: 40px;">t = Built-in topic writer</p> <p style="padding-left: 40px;">w = built-in publication writer</p> <p style="padding-left: 40px;">r = built-in subscription writer</p> <p style="padding-left: 40px;">p (lower case) = built-in participant writer</p> <p style="padding-left: 40px;">m = peer-to-peer participant message writer</p> <p style="padding-left: 40px;">? = unknown writer</p> <p>2 , 3 = two letters that describe the last two bits of the statusInfo inline QoS:</p> <table><tr><th>Bit-1</th><th>Bit-0</th><th>Text</th></tr><tr><td colspan="3">-----</td></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> <p>Where bit 0="Disposed" flag, and bit 1 = Unregistered flag</p> <p>For example, you may see:</p> <p style="padding-left: 40px;">DATA (p [__])</p> <p style="padding-left: 40px;">DATA (p [_D])</p>	Bit-1	Bit-0	Text	-----			0	0	__	0	1	_D	1	0	U_	1	1	UD
Bit-1	Bit-0	Text																	

0	0	__																	
0	1	_D																	
1	0	U_																	
1	1	UD																	
DATA_BATCH	Contains information regarding the values of a batch of application data objects.																		
DATA_FRAG	<p>Contains a fragment of information regarding the value of an application Data-object.</p> <p>For <i>RTI Data Distribution Service 4.2e</i> and higher, a new format is used; captured submessages of the earlier format are displayed as DATA_FRAG_deprecated.</p>																		
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.																		

Table 5.1 RTPS 2.x Submessage Types

Submessage Type	Description
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
INFO_SOURCE	Provides information about the source from which subsequent Entity submessages originated.
INFO_DESTINATION	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. For <i>RTI Data Distribution Service</i> 4.2e and higher, this submessage is not used.
NOKEY_DATA_FRAG	Contains a fragment of information regarding the value of an application data-object that cannot be referenced by a key. For <i>RTI Data Distribution Service</i> 4.2e and higher, this submessage is not used.
PAD	Provides padding to meet any desired memory-alignment requirements.

a. INFO_TS is an abbreviation for INFOTIMESTAMP

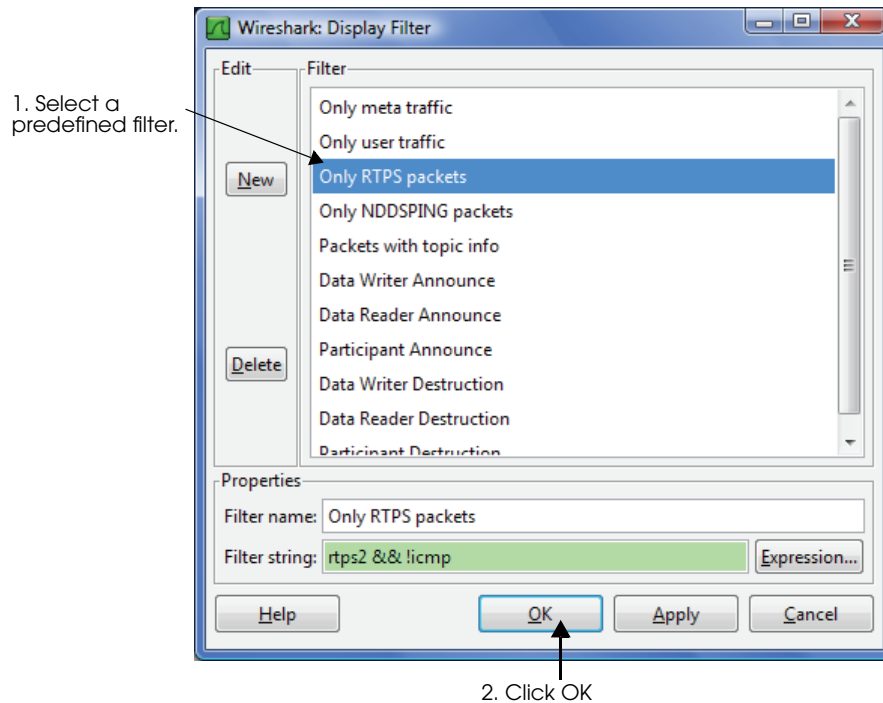
5.2.1 Using a Display Filter

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.”
3. Click **OK** to close the Filter Expression window.

Figure 5.2 Selecting a Display Filter

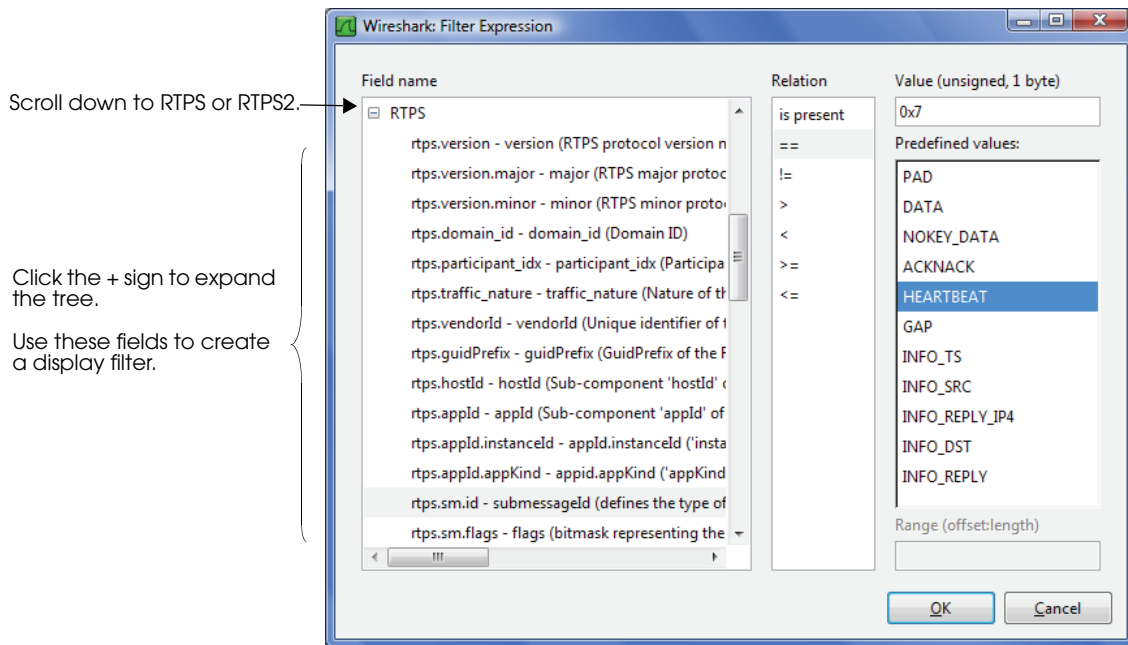


As another example, let's look at how to display only RTPS packets that contain HEARTBEAT submessages.

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 5.3.

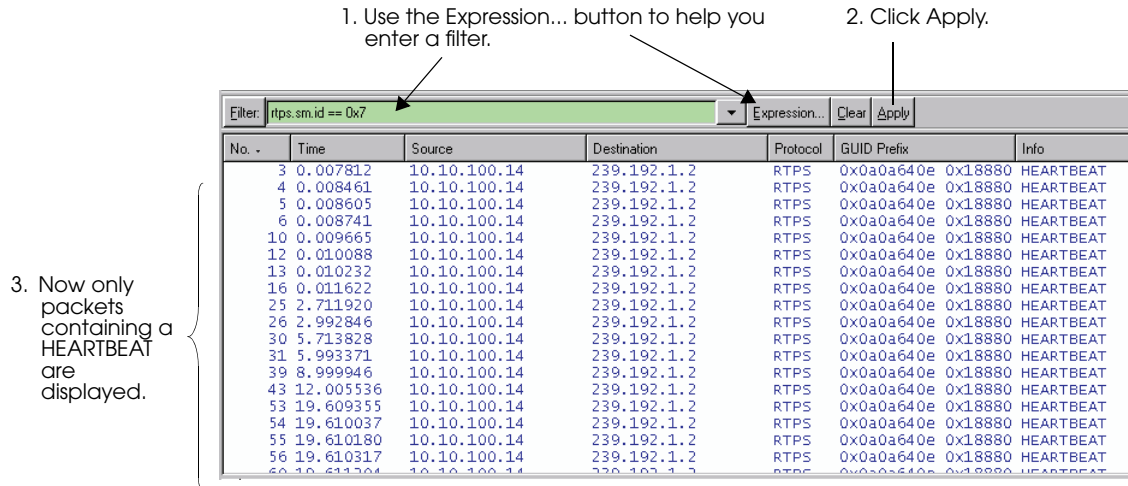
Figure 5.3 **Creating a Display Filter**



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 5.4.

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

Figure 5.4 Filtering by Submessage Type



Displaying RTPS and RTPS2 Messages:

Table 5.2 briefly describes the meaning of each field that can be used in a display filter for RTPS. To display packets for RTPS2, change the prefix from “rtps” to “rtps2.” (These fields can also be used in coloring rules, see Section 5.2.2.) To display both RTPS and RTPS2, rules must include both versions logically OR’ed together. For details on the meaning of these fields, see the *Real-Time Publish-Subscribe Wire Protocol Specification* (see Available Documentation (Section 1.1)).

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
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

Table 5.2 Display-Filter Fields for RTPS Messages

Field	Description
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.sm.flags	Byte representing the submessage flags
rtps.sm.octetsToNextHeader	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.rdentId	Reader entity ID as it appear in a submessage
rtps.sm.rdentId.entityKey	'entityKey' field of the reader entity ID
rtps.sm.rdentId.entityKind	'entityKind' field of the reader entity ID
rtps.sm.wrentId	Writer entity ID as it appear in a submessage
rtps.sm.wrentId.entityKey	'entityKey' field of the writer entity ID
rtps.sm.wrentId.entityKind	'entityKind' field of the writer entity ID
rtps.sm.seqNumber	Writer sequence number

Table 5.2 Display-Filter Fields for RTPS Messages

Field	Description
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

Important: 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 Section 5.2.1). 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. (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 Available Documentation (Section 1.1)). Figure 5.5 shows a sample display.

Figure 5.5 Using Coloring Rules

20	0.020836	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
31	5.993371	10.10.100.14	239.192.1.2	RTPS	0x0a0a640e	0x18880	HEARTBEAT
32	5.993520	10.10.100.14	239.192.1.2	RTPS	0x0a0a640e	0x18880	ACKNACK
33	5.993643	10.10.100.14	239.192.1.2	RTPS	0x0a0a640e	0x18880	ACKNACK
34	5.993743	10.10.100.14	239.192.1.2	RTPS	0x0a0a640e	0x18880	ACKNACK

Coloring rules make is easy to see different types of submessages.

To create a new 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.
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.

Table 5.3 Default Coloring Rules

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

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 RTI Data Distribution Service Applications

RTI's distribution of *Wireshark* includes two files that contain packets captured from *RTI Data Distribution Service* 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`
- ☐ Solaris: `/usr/local/share/wireshark`

- ❑ Windows: <WiresharkHOME>\rti (where <WiresharkHOME> is where *Wireshark* is installed)

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.

Note: these sample traces were taken with a beta version of *RTI Data Distribution Service* (4.0f), which used RTPS 1.1. The protocol used in *RTI Data Distribution Service* 4.2 and higher uses RTPS 2.x. Therefore, the actual sequence of packets exchanged will be different in your traces. These samples are provided only to show you how to read a sequence of RTPS packets.

5.3.1 Analyzing the User Data Sample Trace

Use the **File, Open...** command to open the file, `userDataTrace.pkt` (see [Section 5.3](#) for its location).

The sample file contains a sequence of RTPS packets that illustrate the protocol when two *RTI Data Distribution Service* 4.0 applications use reliable communications to send/receive data.

This scenario involves two hosts, each running one *RTI Data Distribution Service* application.

- ❑ Host 1 (10.10.100.2, named kirkwood) is running an *RTI Data Distribution Service* publishing application, App1.
- ❑ Host 2 (10.10.50.247, named kootenay) is running an *RTI Data Distribution Service* subscribing application, App2.
- ❑ The QoS for the writer and 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 (kirkwood) *after* the discovery process completed, using the following capture filter:

```
udp and src or dst kootenay
```

Figure 5.6 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

Table 5.4 and Figure 5.7 describe the trace.

Figure 5.6 User Data Sample Packets

No. -	Time	Source	Destination	Protocol	GUID Prefix	Info
1	0.000000	10.10.100.2	10.10.50.247	RTPS	0x0a0a6402	0x09200 PAD, NOKEY_DA
2	1.124057	10.10.100.2	10.10.50.247	RTPS	0x0a0a6402	0x09200 HEARTBEAT
3	1.124179	10.10.50.247	10.10.100.2	RTPS	0x0a0a32f7	0x63cf0 ACKNACK
4	3.999752	10.10.100.2	10.10.50.247	RTPS	0x0a0a6402	0x09200 PAD, NOKEY_DA
5	4.123881	10.10.100.2	10.10.50.247	RTPS	0x0a0a6402	0x09200 HEARTBEAT
6	4.124773	10.10.50.247	10.10.100.2	RTPS	0x0a0a32f7	0x63cf0 ACKNACK
7	7.999506	10.10.100.2	10.10.50.247	RTPS	0x0a0a6402	0x09200 PAD, NOKEY_DA
8	10.123530	10.10.100.2	10.10.50.247	RTPS	0x0a0a6402	0x09200 HEARTBEAT
9	10.123647	10.10.50.247	10.10.100.2	RTPS	0x0a0a32f7	0x63cf0 ACKNACK

Table 5.4 Analysis of User Data Sample Trace

Direction	Packet #	Description
App1 → App2	1	Data packet sent to the reader (NOKEY_DATA submessage). Packet has sequence number = 60 (expand the protocol tree in the Packet Details pane and check the writerSeqNumber value, as seen in Figure 5.8).
	2	HEARTBEAT from writer to reader
App1 ← App2	3	ACKNACK to acknowledge all data packets up to (but not including) sequence number 61 (expand the protocol tree in the Packet Details pane and check the 'readerSNState.bitmapBase' value).
App1 → App2	4	Another data packet (sequence number 61).
	5	HEARTBEAT from writer to reader.
App1 ← App2	6	Acknowledges packet #4.
App1 → App2	7	Another data packet (sequence number 62).
	8	HEARTBEAT from writer to reader.
App1 ← App2	9	ACKNACK to acknowledge packet #7.

Figure 5.7 User Data Sample Packet Flow

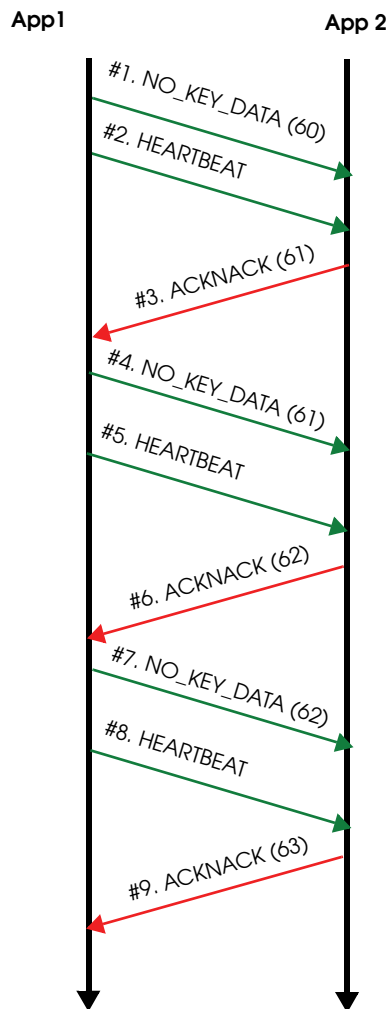


Figure 5.8 Examining Packet Details

Select a packet in the Packet List

No. -	Time	Source	Destination	Protocol	GUID Prefix	Info
1	0.000000	10.10.100.2	10.10.50.247	RTPS	0x0a0a6402 0x09200	PAD, NOKEY_DATA
2	1.124057	10.10.100.2	10.10.50.247	RTPS	0x0a0a6402 0x09200	HEARTBEAT
3	1.124179	10.10.50.247	10.10.100.2	RTPS	0x0a0a32f7 0x63cf0	ACKNACK
4	3.999752	10.10.100.2	10.10.50.247	RTPS	0x0a0a6402 0x09200	PAD, NOKEY_DATA
5	4.123881	10.10.100.2	10.10.50.247	RTPS	0x0a0a6402 0x09200	HEARTBEAT
6	4.124773	10.10.50.247	10.10.100.2	RTPS	0x0a0a32f7 0x63cf0	ACKNACK
7	7.999506	10.10.100.2	10.10.50.247	RTPS	0x0a0a6402 0x09200	PAD, NOKEY_DATA
8	10.123530	10.10.100.2	10.10.50.247	RTPS	0x0a0a6402 0x09200	HEARTBEAT
9	10.123647	10.10.50.247	10.10.100.2	RTPS	0x0a0a32f7 0x63cf0	ACKNACK

Frame 1 (90 bytes on wire, 90 bytes captured)

- ⊞ Ethernet II, Src: Intel_0b:d2:a7 (00:11:11:0b:d2:a7), Dst: Intel_09:38:dc (00:11:11:09:38:dc)
- ⊞ Internet Protocol, Src: 10.10.100.2 (10.10.100.2), Dst: 10.10.50.247 (10.10.50.247)
- ⊞ User Datagram Protocol, Src Port: 1190 (1190), Dst Port: 8273 (8273)
- ⊞ Real-Time Publish-Subscribe wire Protocol
 - ⊞ Protocol version: 1.1
 - major: 1
 - minor: 1
 - vendor: Real-Time Innovations, Inc.
 - ⊞ guidPrefix=0a0a6402 09200001 { hostId=0a0a6402, appId=09200001 (ManagedApplication: 092000) }
 - ⊞ Default port mapping: domainId=87, participantIdx=0, nature=UNICAST_USERTRAFFIC
 - ⊞ Submessage: PAD
 - submessageId: PAD (0x01)
 - ⊞ Flags: 0x01 (_ _ _ _ _ E)
 - octetsToNextHeader: 0
 - ⊞ Submessage: NOKEY_DATA
 - submessageId: NOKEY_DATA (0x03)
 - ⊞ Flags: 0x01 (_ _ _ _ _ E)
 - octetsToNextHeader: 24
 - ⊞ readerEntityId: ENTITYID_UNKNOWN (0x00000000)
 - ⊞ writerEntityId: 0x00000303 (Application-defined writer (no key): 0x000003)
 - ⊞ writerSeqNumber: 60
 - ⊞ serializedData: 3B000000398E8142

Expand the submessage details to see the sequence number and other details.

5.3.2 Analyzing the Discovery Data Sample Trace

Use the **File, Open...** command to open the file, `discoveryTrace.pkt` (see [Section 5.3](#) for its location).

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

This scenario involves two hosts, each running one *RTI Data Distribution Service* application.

- ❑ Host 1 (10.10.100.2) is running an *RTI Data Distribution Service* publishing application, App1.
- ❑ Host 2 (10.10.50.247) is running an *RTI Data Distribution Service* subscribing application, App2.
- ❑ Both applications have a maximum participant index of 1 and have each other in their `initial_peer_list`.
- ❑ All QoS are at default values, including the use of automatic discovery via the default UDPv4 transport.

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 5.9](#) shows the packets captured by *Wireshark*.

Let's walk through the RTPS packets to understand what occurred in this sequence. [Table 5.5](#) 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).

Figure 5.9 Discovery Data Sample File

No. -	Time	Source	Destination	Protocol	GUID Prefix	Info
1	0.000000	10.10.100.2	10.10.50.247	UDP	Unknown	Source port: 3700 Dest
2	0.000199	10.10.100.2	10.10.50.247	RTPS	0x0a0a6402 0x0c180	DATA
3	0.000230	10.10.100.2	10.10.50.247	RTPS	0x0a0a6402 0x0c180	DATA
4	6.041247	10.10.50.247	10.10.100.2	UDP	Unknown	Source port: 36816 Des
5	6.041369	10.10.50.247	10.10.100.2	RTPS	0x0a0a32f7 0x46000	DATA
6	6.041385	10.10.50.247	10.10.100.2	RTPS	0x0a0a32f7 0x46000	DATA
7	6.041981	10.10.100.2	10.10.50.247	RTPS	0x0a0a6402 0x0c180	DATA
8	6.042019	10.10.100.2	10.10.50.247	RTPS	0x0a0a6402 0x0c180	DATA
9	6.042100	10.10.100.2	10.10.50.247	RTPS	0x0a0a6402 0x0c180	HEARTBEAT
10	6.042198	10.10.100.2	10.10.50.247	RTPS	0x0a0a6402 0x0c180	HEARTBEAT
11	6.042302	10.10.100.2	10.10.50.247	RTPS	0x0a0a6402 0x0c180	ACKNACK
12	6.042373	10.10.100.2	10.10.50.247	RTPS	0x0a0a6402 0x0c180	ACKNACK
13	6.042492	10.10.100.2	10.10.50.247	RTPS	0x0a0a6402 0x0c180	DATA
14	6.042513	10.10.100.2	10.10.50.247	RTPS	0x0a0a6402 0x0c180	DATA
15	6.042878	10.10.50.247	10.10.100.2	RTPS	0x0a0a32f7 0x46000	DATA
16	6.042896	10.10.50.247	10.10.100.2	RTPS	0x0a0a32f7 0x46000	DATA
17	6.042981	10.10.50.247	10.10.100.2	RTPS	0x0a0a32f7 0x46000	HEARTBEAT
18	6.043024	10.10.50.247	10.10.100.2	RTPS	0x0a0a32f7 0x46000	HEARTBEAT
19	6.043076	10.10.50.247	10.10.100.2	RTPS	0x0a0a32f7 0x46000	ACKNACK
20	6.043174	10.10.50.247	10.10.100.2	RTPS	0x0a0a32f7 0x46000	ACKNACK
21	6.043219	10.10.100.2	10.10.50.247	RTPS	0x0a0a6402 0x0c180	ACKNACK
22	6.043330	10.10.100.2	10.10.50.247	RTPS	0x0a0a6402 0x0c180	HEARTBEAT
23	6.043368	10.10.50.247	10.10.100.2	RTPS	0x0a0a32f7 0x46000	DATA
24	6.043380	10.10.50.247	10.10.100.2	RTPS	0x0a0a32f7 0x46000	DATA
25	6.043389	10.10.100.2	10.10.50.247	RTPS	0x0a0a6402 0x0c180	HEARTBEAT
26	6.043457	10.10.50.247	10.10.100.2	RTPS	0x0a0a32f7 0x46000	ACKNACK
27	6.043514	10.10.50.247	10.10.100.2	RTPS	0x0a0a32f7 0x46000	HEARTBEAT
28	6.043551	10.10.50.247	10.10.100.2	RTPS	0x0a0a32f7 0x46000	HEARTBEAT
29	6.043640	10.10.50.247	10.10.100.2	RTPS	0x0a0a32f7 0x46000	DATA(r+), HEARTBEAT
30	6.043668	10.10.100.2	10.10.50.247	RTPS	0x0a0a6402 0x0c180	DATA(w+), HEARTBEAT
31	6.043670	10.10.50.247	10.10.100.2	RTPS	0x0a0a32f7 0x46000	ACKNACK
32	6.043691	10.10.100.2	10.10.50.247	RTPS	0x0a0a6402 0x0c180	ACKNACK
33	6.043698	10.10.50.247	10.10.100.2	RTPS	0x0a0a32f7 0x46000	ACKNACK
34	6.043759	10.10.100.2	10.10.50.247	RTPS	0x0a0a6402 0x0c180	ACKNACK
35	6.043794	10.10.50.247	10.10.100.2	RTPS	0x0a0a32f7 0x46000	ACKNACK
36	6.043829	10.10.50.247	10.10.100.2	RTPS	0x0a0a32f7 0x46000	DATA(r+), HEARTBEAT
37	6.043973	10.10.100.2	10.10.50.247	RTPS	0x0a0a6402 0x0c180	ACKNACK
38	6.044072	10.10.50.247	10.10.100.2	RTPS	0x0a0a32f7 0x46000	DATA
39	6.044087	10.10.50.247	10.10.100.2	RTPS	0x0a0a32f7 0x46000	DATA
40	6.044098	10.10.100.2	10.10.50.247	RTPS	0x0a0a6402 0x0c180	DATA(w+), HEARTBEAT
41	6.044132	10.10.100.2	10.10.50.247	RTPS	0x0a0a6402 0x0c180	ACKNACK
42	6.044198	10.10.50.247	10.10.100.2	RTPS	0x0a0a32f7 0x46000	ACKNACK
43	6.044278	10.10.100.2	10.10.50.247	RTPS	0x0a0a6402 0x0c180	DATA
44	6.044331	10.10.100.2	10.10.50.247	RTPS	0x0a0a6402 0x0c180	DATA
45	7.989018	10.10.100.2	10.10.50.247	RTPS	0x0a0a6402 0x0c180	PAD, NOKEY_DATA
46	11.988696	10.10.100.2	10.10.50.247	RTPS	0x0a0a6402 0x0c180	PAD, NOKEY_DATA
47	15.988537	10.10.100.2	10.10.50.247	RTPS	0x0a0a6402 0x0c180	PAD, NOKEY_DATA
48	19.988296	10.10.100.2	10.10.50.247	RTPS	0x0a0a6402 0x0c180	PAD, NOKEY_DATA

Table 5.5 Analysis of Sample File's Packets

Direction	Packet #	Description
App1 → App2	1 - 3	<p>When the writer participant starts, <i>RTI Data Distribution Service</i> announces the creation of a new participant to all potential participants in the initial_peer_list.</p> <p>Potential participants are initially calculated as: for each peer in initial_peer_list, peer/participant(i), where i ≤ maximum participant index.</p> <p>Since the participant's maximum participant index is 1 and initial_peer_list contains only 10.10.50.247, the potential participant list is {10.10.50.247/participant(0), 10.10.50.247/participant(1)}.</p> <p>Since each participant gets its own receive locator, we send separate (but identical) packets to each potential participant listening on its own locator, as shown in packets #2 and #3.</p> <p>#1 is sent by the <i>RTI Data Distribution Service</i> pluggable transport module to see if the specified locator is a reachable destination. Since there is no other <i>RTI Data Distribution Service</i> application in the system on the same domain index, there will be no response to packets #2 and #3.</p>
App1 ← App2	4 - 6	<p>Similar to the writer participant, when the reader participant (on participant index 1) starts, <i>RTI Data Distribution Service</i> announces the new participant.</p>
App1 → App2	7,8	<p>When the writer participant learns about the new participant, it announces itself again to the new participant (and all other potential participants).</p>
	9, 10	<p>Writer participant uses HEARTBEATS to tell 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.</p> <p>To be strict, this sequence number does not imply that there is 1 writer, but that the writer participant has made 1 change to the internal participant meta group. This number would get incremented if the writer was changed or deleted.</p>
	11, 12	<p>ACKNACKs from writer participant asking the reader participant about its readers and writer.</p>
	13, 14	<p>Repeat announcements about the writer participant. Repeat announcements reduce the chance that the newly created reader participant will drop the reply from the writer participant.</p>

Table 5.5 Analysis of Sample File's Packets

Direction	Packet #	Description
App1 ← App2	15, 16	Reader participant learns about the writer participant and announces itself to the world again.
	17, 18	Reader participant tells the writer participant how many readers and writers it has.
	19, 20	Reader participant asks the writer to update the reader and writer meta DATA.
App1 → App2	21	Writer, likewise, asks the reader to send updates of its reader meta DATA.
	22	Writer responds to #19 and tells the reader that it doesn't have any readers.
App1 ← App2	23, 24	Reader refreshes its participant declaration to the world.
App1 → App2	25	Writer responds to #20, saying it has 1 writer.
App1 ← App2	26	Reader responds, asking for the writer meta DATA it hasn't received.
	27 - 29	Reader participant repeats its response to the query in #21 by sending the meta DATA for its reader.
	30	Writer participant sends the meta DATA for its writer. This is a response to #26.
App1 ← App2	31	Reader participant acknowledges the receipt of the meta DATA. This is a response to #22.
App1 → App2	32	Reader and writer participants exchange meta information about what meta DATA they have received from each other. #32 is a response to #27 (not #29).
App1 ← App2	33	#33 is a response to #25.
App1 → App2	34	
App1 ← App2	35	Writer participant acknowledges receipt of meta DATA. This is a response to #30.
	36	Reader participant send meta DATA for its reader. This is a response to #32.
App1 → App2	37	Writer participant acknowledges receipt of this meta DATA. This is a response to #29.
App1 ← App2	38, 39	Reader participant announces itself to the world.
App1 → App2	40	Writer responds to reader participant's query in #33. Note that this is redundant with #30 due to cross talk.
	41	Redundant acknowledgements.
App1 ← App2	42	Redundant acknowledgements.
App1 → App2	43 - 44	Writer reannounces itself to the world.
	45 - 48	Writer finally begins sending user data.

Chapter 6 Practical Uses with RTI Applications

This chapter offers a few suggestions on how *Wireshark* can be used during *RTI Data Distribution Service* application development:

- ❑ [Debugging Discovery Problems \(Section 6.1\)](#)
- ❑ [Visualizing Your System \(Section 6.2\)](#)
- ❑ [Providing Information to RTI Support \(Section 6.3\)](#)

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. *RTI Data Distribution Service* 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. *RTI Data Distribution Service* uses separate built-in objects to announce and discover readers vs. writers, so it's important to check the *writer-EntityId* 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 DDS entities (e.g. participant, topic, reader, writer), 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, *RTI Data Distribution Service* 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 Data Distribution Service User's Manual* or online documentation for information about the Liveliness QosPolicy.)

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

6.2 Visualizing Your System

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

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

As Figures 6.1 through 6.3 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
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
10.10.1.81	138	10.10.255.255	138	3	804	3	804	0	0

Figure 6.2 I/O Graph

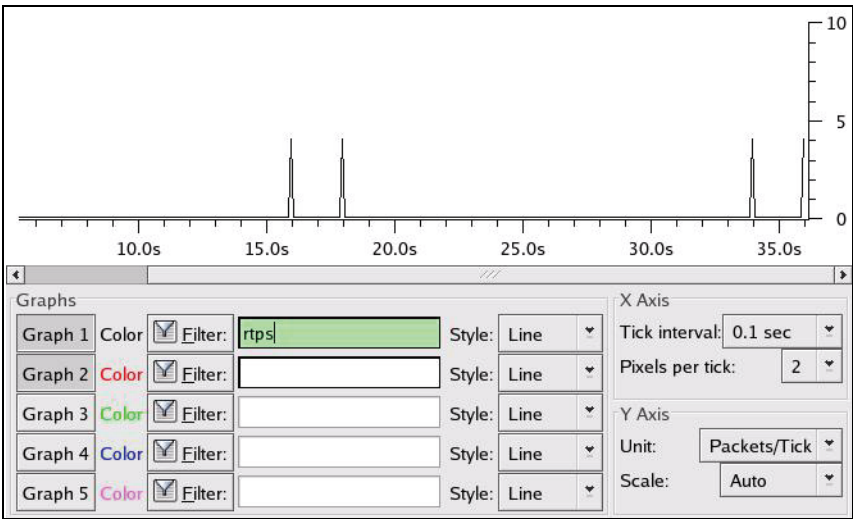


Figure 6.3 Protocol Hierarchy

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

6.3 Providing Information to RTI Support

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

See the *RTI Data Distribution Service User's Manual's* Troubleshooting chapter for more information on setting verbosity.

