Date: February 2018

TCP/IP Platform-Specific Model (PSM) for the DDS Real-Time Publish Subscribe (DDS-RTPS) Protocol

Version 1.0 Draft

OMG Document Number mars/2018-02-15

Normative Reference: http://www.omg.org/spec/DDSI-RTPS-TCP/1.0

Specification Template: omg/16-05-01

Copyright © 2018, Real-Time Innovations, Inc.

Copyright © 2018, Twin Oaks Computing, Inc.  
Copyright © 2018, Object Management Group, Inc.

USE OF SPECIFICATION – TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

LICENSES

The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.

Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

GENERAL USE RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--without permission of the copyright owner.

DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE. IN NO EVENT SHALL THE OBJECT MANAGEMENT GROUP OR ANY OF THE COMPANIES LISTED ABOVE BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The entire risk as to the quality and performance of software developed using this specification is borne by you. This disclaimer of warranty constitutes an essential part of the license granted to you to use this specification.

RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c) (1) (ii) of The Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clauses at 48 C.F.R. 52.227-19 or as specified in 48 C.F.R. 227-7202-2 of the DoD F.A.R. Supplement and its successors, or as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors, as applicable. The specification copyright owners are as indicated above and may be contacted through the Object Management Group, 140 Kendrick Street, Needham, MA 02494, U.S.A.

TRADEMARKS

CORBA®, CORBA logos®, FIBO®, Financial Industry Business Ontology®, FINANCIAL INSTRUMENT GLOBAL IDENTIFIER®, IIOP®, IMM®, Model Driven Architecture®, MDA®, Object Management Group®, OMG®, OMG Logo®, SoaML®, SOAML®, SysML®, UAF®, Unified Modeling Language®, UML®, UML Cube Logo®, VSIPL®, and XMI® are registered trademarks of the Object Management Group, Inc.

For a complete list of trademarks, see: http://www.omg.org/legal/tm\_list.htm. All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

COMPLIANCE

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials.

Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.

OMG’s Issue Reporting Procedure

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page http://www.omg.org, under Documents, Report a Bug/Issue (http://issues.omg.org/issues/create-new-issue).

Table of Contents

[0 Response Details 1](#__RefHeading__2838_475089943)

[0.1 OMG Response Details 1](#__RefHeading__102572_465144765)

[0.2 Copyright Waiver 1](#__RefHeading___Toc28391_925963519)

[0.3 Contacts 1](#__RefHeading___Toc28393_925963519)

[0.4 Problem Statement 1](#__RefHeading___Toc28395_925963519)

[0.5 Overview of this Specification 1](#__RefHeading___Toc28397_925963519)

[0.6 Mapping to RFP Requirements 1](#__RefHeading___Toc28399_925963519)

[0.7 Responses to RFP Issues to Be Discussed 3](#__RefHeading___Toc28401_925963519)

[1 Scope 5](#__RefHeading__1546_1593348256)

[2 Conformance 5](#__RefHeading__1548_1593348256)

[3 Normative References 5](#__RefHeading__1747_2093095907)

[4 Terms and Definitions 5](#__RefHeading__1552_1593348256)

[5 Symbols 6](#__RefHeading__1554_1593348256)

[6 Additional Information 6](#__RefHeading__1556_1593348256)

[6.1 Changes to Adopted OMG Specifications 6](#__RefHeading__1558_1593348256)

[6.2 Acknowledgements 6](#__RefHeading__1560_1593348256)

[7 Overview 7](#__RefHeading__1564_1593348256)

[7.1 Introduction 7](#__RefHeading__105252_1593348256)

[7.2 The RTPS Platform Independent Model (PIM) in a Nutshell (non-normative) 7](#__RefHeading__2840_475089943)

[7.3 Discussion of TCP/IP Semantics and Relevant Issues (non-normative) 7](#__RefHeading___Toc7691_346059697)

[7.3.1 Asymmetric Connection Establishment 7](#__RefHeading__2842_475089943)

[7.3.2 Stream-Oriented Protocol 7](#__RefHeading___Toc7693_346059697)

[7.3.3 Reliable Communication 8](#__RefHeading___Toc7695_346059697)

[7.3.4 Head-Of-Line (HOL) Blocking 8](#__RefHeading___Toc7697_346059697)

[7.3.5 Multicast Support 8](#__RefHeading___Toc7699_346059697)

[7.4 Deployment Scenarios 8](#__RefHeading___Toc7701_346059697)

[7.4.1 Direct Communication 8](#__RefHeading___Toc7703_346059697)

[7.4.2 Client Behind a NAT 9](#__RefHeading___Toc7705_346059697)

[7.5 Technical Issues in using TCP/IP for an RTPS PSM 10](#__RefHeading___Toc8103_346059697)

[8 DDS-RTPS Platform Specific Model (PSM) for TCP/IP 11](#__RefHeading___Toc8105_346059697)

[8.1 Introduction 11](#__RefHeading___Toc8107_346059697)

[8.2 Notational Conventions 11](#__RefHeading___Toc8109_346059697)

[8.3 Mapping of the RTPS Types 11](#__RefHeading___Toc8111_346059697)

[8.4 Mapping of RTPS Messages 12](#__RefHeading___Toc8113_346059697)

[8.4.1 Overall Structure 12](#__RefHeading___Toc8115_346059697)

[8.4.2 Mapping of PIM SubmessageElements 12](#__RefHeading___Toc8117_346059697)

[8.4.3 Additional SubmessageElements 13](#__RefHeading___Toc8119_346059697)

[8.4.4 Mapping of the RTPS Header 13](#__RefHeading___Toc8121_346059697)

[8.4.5 Mapping of the RTPS Submessages 14](#__RefHeading___Toc8123_346059697)

[8.5 RTPS Message Encapsulation 14](#__RefHeading___Toc8125_346059697)

[8.6 Mapping of the RTPS Protocol 14](#__RefHeading___Toc8127_346059697)

[8.6.1 Default Locators 14](#__RefHeading___Toc8129_346059697)

[8.6.2 Data Representation for the Built-in Endpoints 14](#__RefHeading___Toc8131_346059697)

[8.6.3 ParameterId Definitions Used to Represent In-line QoS 14](#__RefHeading___Toc8133_346059697)

[9 RTPS TCP Control Protocol (TO BE DISCUSSED -- needs agreement) 15](#__RefHeading__1580_1593348256)

[9.1 Overview 15](#__RefHeading__13302_294284811)

[9.2 Control Protocol Behavior 15](#__RefHeading___Toc8135_346059697)

[9.3 Control Protocol Messages 16](#__RefHeading___Toc8137_346059697)

[9.3.1 Control Protocol Message Header 16](#__RefHeading___Toc8139_346059697)

[9.3.2 Control Message Data 18](#__RefHeading___Toc10634_346059697)

[10 Use of the RTPS TCP/IP PSM (non-normative) 23](#__RefHeading___Toc15911_346059697)

[10.1 Overview 23](#__RefHeading___Toc15913_346059697)

[10.2 Use of RTPS TCP PSM/IP PSM in Deployment Scenarios 23](#__RefHeading___Toc15915_346059697)

[10.2.1 LAN Deployment 23](#__RefHeading___Toc15917_346059697)

[10.2.2 WAN Deployments 24](#__RefHeading___Toc15919_346059697)

Table of Figures

Figure 7.1: Direct Communication 9

Figure 7.2: Client Behind a NAT 9

Figure 7.3: Client and Server Behind NATs—TCP for WAN only 10

Figure 7.4: WAN Deployment Behind NATs—TCP for LAN and WAN 10

Figure 9.1: Control Protocol Overview 16

Figure 10.1: LAN Deployment Detailed Example 24

Table of Tables

Table 0.1: Mandatory Requirements 2

Table 0.2: Non-Mandatory Requirements 3

Table 0.3: Issues to be Discussed 3

Table 8.1: PSM mapping of the value types that appear on the wire 11

Table 9.1: Control message kinds and their 8-bit corresponding value 17

Preface

OMG

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable, and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies, and academia.

OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG’s specifications implement the Model Driven Architecture® (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG’s specifications include: UML® (Unified Modeling Language®); CORBA® (Common Object Request Broker Architecture); CWM™ (Common Warehouse Metamodel™); and industry-specific standards for dozens of vertical markets.

More information on the OMG is available at http://www.omg.org/.

OMG Specifications

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. All OMG Specifications are available from the OMG website at:

http://www.omg.org/spec

Specifications are organized by the following categories:

Business Modeling Specifications

Middleware Specifications

* CORBA/IIOP
* Data Distribution Services
* Specialized CORBA

IDL/Language Mapping Specifications

Modeling and Metadata Specifications

* UML, MOF, CWM, XMI
* UML Profiles

Modernization Specifications

Platform Independent Model (PIM), Platform Specific Model (PSM), Interface Specifications

* CORBAServices
* CORBAFacilities

OMG Domain Specifications

CORBA Embedded Intelligence Specifications

CORBA Security Specifications

Signal and Image Processing Specifications

All of OMG’s formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format, may be obtained from the Specifications Catalog cited above or by contacting the Object Management Group, Inc. at:

OMG Headquarters  
109 Highland Avenue  
Needham, MA 02494  
USA

Tel: +1-781-444-0404  
Fax: +1-781-444-0320  
Email: pubs@omg.org

Certain OMG specifications are also available as ISO standards. Please consult http://www.iso.org

Typographical Conventions

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or section headings where no distinction is necessary.

Times/Times New Roman/Liberation Serif – 10 pt.: Standard body text

Helvetica/Arial – 10 pt. Bold: OMG Interface Definition Language (OMG IDL) and syntax elements.

Courier – 10 pt. Bold: Programming language elements.

Helvetica/Arial – 10 pt: Exceptions

NOTE: Terms that appear in italics are defined in the glossary. Italic text also represents the name of a document, specification, or other publication.

Issues

The reader is encouraged to report any technical or editing issues/problems with this specification via the report form at:

http://issues.omg.org/issues/create-new-issue

This page intentionally left blank.

# Response Details

## OMG Response Details

This specification is submitted in response to the TCP/IP Platform-Specific Model (PSM) for the DDS Real-Time Publish Subscribe (RTPS) Protocol RFP issued in September 2017 with OMG document number mars/2017-09-03.

## Copyright Waiver

“Each of the entities listed above: (i) grants to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version, and (ii) grants to each member of the OMG a nonexclusive, royalty-free, paid up, worldwide license to make up to fifty (50) copies of this document for internal review purposes only and not for distribution, and (iii) has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used any OMG specification that may be based hereon or having conformed any computer software to such specification.”

## Contacts

* Real-Time Innovations, Inc. Fernando Garcia-Aranda fgarcia AT rti.com
* Twin Oaks Computing, Inc. Clark Tucker ctucker AT twinoakscomputing.com

## Problem Statement

The DDS Real-time Publish-Subscribe wire protocol (DDSI-RTPS) specification defines a set of requirements for a wire protocol suitable for the Data Distribution Service (DDS). Primary considerations in the design of the RTPS wire protocol are: performance, configurability (tuning quality-of-service), fault-tolerance (no single points of failure), extensibility (support new transports), plug-and-play connectivity (automatic discovery), modularity, scalability, and type safety.

RTPS imposes very little requirements on the underlying transport: a connectionless service capable of sending packets best-effort is sufficient. A connection-oriented protocol can be used but is not required. The mechanisms of the underlying protocol map to the generalized notions of the RTPS Platform Independent Model (PIM).

The original DDSI-RTPS specification defined a Platform Specific Model (PSM) built upon the User Datagram Protocol (UDP) because of its simplicity, universal availability, best-effort and connectionless capabilities, predictable behavior, scalability, and multicast support.

However, some DDS systems would benefit from an RTPS PSM built upon the Transmission Control Protocol (TCP). Among other scenarios, a TCP PSM would be better suited for communication through firewalls, where often UDP traffic is filtered; could leverage existing TCP-based load-balancing infrastructure; and would allow DDS to be deployed in some applications where governance mandates TCP exclusively. Therefore, the goal of this RFP is to meet the requirements set forth by the RTPS PIM with minimum possible overhead using TCP.

## Overview of this Specification

## Mapping to RFP Requirements

Table 0.1 lists the Mandatory Requirements in the RFP and how this submission addresses them.

Table 0.1: Mandatory Requirements

| Requirment | Description | How Is The Requirement Addressed |
| --- | --- | --- |
| 6.5.1.1 | Proposals shall provide a PSM, derived from the DDSI-RTPS PIM, targeting TCP. |  |
| 6.5.1.2 | The proposed PSM shall implement the DDSI-RTPS PIM in its entirety. |  |
| 6.5.1.3 | The proposed PSM shall not introduce dependencies on other communication middleware technologies. |  |
| 6.5.1.4 | Proposals shall reuse the sub messages defined in the DDSI-RTPS PIM. |  |
| 6.5.1.5 | The proposed PSM shall not change or extend the DDS API. |  |
| 6.5.2.1 | The proposed PSM shall support deployments where participants behaving as a client and a server communicate directly over TCP without intervening NATs. |  |
| 6.5.2.2 | The proposed PSM shall support deployments where the participant behaving as a client is behind a NAT. |  |
| 6.5.2.3 | The proposed PSM shall support deployments where the participant behaving as a server is behind a NAT. |  |
| 6.5.2.4 | The proposed PSM shall support deployments where the participants behaving as a client and a server are both behind a NAT. |  |
| 6.5.2.5 | The proposed PSM shall support deployments where participants can communicate using both UDP and TCP. |  |
| 6.5.3.1 | The proposed PSM shall support communication over a single TCP connection. |  |
| 6.5.3.2 | The proposed PSM shall specify the behavior when two participants attempt to establish a TCP connection with one another. |  |
| 6.5.3.3 | The proposed PSM shall specify a mechanism to handle TCP connection drops without affecting DDS communications. |  |

Table 0.2 lists the Non-Mandatory Requirements in the RFP and how this submission addresses them.

Table 0.2: Non-Mandatory Requirements

| Requirment | Description | How Is The Requirement Addressed |
| --- | --- | --- |
| 6.6.1.1 | The proposed PSM may provide security extensions compatible with [DDS-SECURITY]. |  |
| 6.6.2.1 | The proposed PSM may support communication over different TCP connections (e.g., different connections for sending user data and sending discovery data). |  |
| 6.6.2.2 | The proposed PSM may support communication over different TCP connections through load balancers. |  |
| 6.6.2.3 | The proposed PSM may provide a mechanism to exchange discovery and data traffic to/from multiple participants sharing a TCP connection. |  |

## Responses to RFP Issues to Be Discussed

Table 0.3 lists the Issues to Discussed in the RFP and how this submission addresses them.

Table 0.3: Issues to be Discussed

| Issue | Description | How Is the Issue Discussed |
| --- | --- | --- |
| 6.7.1.1 | Proposals shall discuss and provide the rationale for any additions to the DDSI-RTPS PIM. |  |
| 6.7.2.1 | Proposals shall mirror the chapter structure of the DDSI-RTPS UDP PSM and discuss any additions to it. |  |

This page intentionally left blank

# Scope

The goal of this RFC is to standardize a TCP/IP transport mapping for the Real-Time Publish Subscribe DDS interoperability wire protocol (DDS-RTPS) specification. Its purpose is to ensure that applications based on different implementations of RTPS can interoperate using the TCP/IP protocol.

# Conformance

This specification contains no independent conformance points.

# Normative References

The following normative documents contain provisions which, through reference in this text, constitute provisions of this specification. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply.

[DDS] OMG, Data Distribution Service, Version 1.4, http://www.omg.org/spec/DDS/1.4.

[DDS-RTPS] OMG, The Real-Time Publish-Subscribe (RTPS) DDS Interoperability Protocol, Version 2.2, http://www.omg.org/spec/DDSI-RTPS/2.2.

[DDS-SECURITY] OMG, DDS Security Specification, Version 1.1, http://www.omg.org/members/spec/DDS-SECURITY/1.1.

[DDS-XTYPES] OMG, Extensible and Dynamic Topic Types for DDS, Version 1.2, http://www.omg.org/spec/DDS-XTypes/1.2.

[IDL] OMG, Interface Definition Language (IDL), Version 4.2., http://www.omg.org/spec/IDL/4.2.

[TCP] Internet Engineering Task Force (IETF), Transport Control Protocol, https://tools.ietf.org/html/rfc793.

# Terms and Definitions

For the purposes of this specification, the following terms and definitions apply.

Data Distribution Service (DDS)

An OMG distributed data communications specification that allows Quality of Service (QoS) policies to be specified for data timeliness and reliability.

Network Address Translation (NAT)

Network address translation (NAT) is the process of modifying IP address information in IP packet headers while in transit across a traffic routing device.

Platform-Independent Model (PIM)

An abstract definition of a facility, often expressed with the aid of formal or semi-formal modeling languages such as OMG UML, which does not depend on any particular implementation technology.

Platform-Specific Model (PSM)

A concrete definition of a facility—typically based on a corresponding PIM—in which all implementation-specific dependencies have been resolved.

Transmission Control Protocol (TCP)

The Transmission Control Protocol (TCP) is one of the core protocols of the Internet Protocol Suite. TCP provides reliable ordered delivery of a stream of bytes.

# Symbols

The following acronyms are used in this specification:

* CDR – Common Data Representation
* DDS – Data Distribution Service
* IDL – Interface Definition Language
* PIM – Platform-Independent Model
* PSM – Platform-Specific Model
* RTPS – Real-Time Publish-Subscribe
* OMG – Object Management Group
* XTYPES – eXtensible and dynamic topic TYPES (for DDS)

# Additional Information

## Changes to Adopted OMG Specifications

There are no changes to adopted OMG specifications.

## Acknowledgements

The following companies submitted this specification:

* Real-Time Innovations, Inc.
* Twin Oaks Computing, Inc.

# Overview

## Introduction

The DDS Real-time Publish-Subscribe wire protocol (DDS-RTPS) specification [DDS-RTPS] uses a Model-Driven Architecture (MDA) to describe a Platform Independent Model (PIM) of an interoperability wire protocol, and a Platform Specific Model (PSM) that allows conforming implementations to interoperate over UDP/IP.

The purpose of this specification is to define a standard TCP/IP-based PSM so that implementations conforming to this specifications can alternatively interoperate over TCP/IP. This specification does not modify or deprecate UDP-based interoperability PSM or the RTPS PIM.

## The RTPS Platform Independent Model (PIM) in a Nutshell (non-normative)

The RTPS specification [DDS-RTPS] specifies requirements for a wire protocol suitable for the Data Distribution Service [DDS]. Primary considerations in the design of RTPS wire protocol are: performance, configurability (tuning quality-of-service), fault-tolerance (no single points of failure), extensibility (support new transports), plug-and-play connectivity (automatic discovery), modularity, scalability, and type safety.

RTPS imposes very little requirements on the underlying transport. A connectionless service capable of sending packets in best-effort is sufficient. A connection-oriented protocol can be used but is not required. The RTPS transport model also defines generalized notions of unicast address/port. The mechanisms of the underlying protocol map to the generalized notions of RTPS PIM.

## Discussion of TCP/IP Semantics and Relevant Issues (non-normative)

The Transmission Control Protocol (TCP) is one of the core protocols of the Internet protocol suite [TCP]. It provides reliable, ordered, stream-oriented delivery of bytes from the sender to the receiver. TCP/IP is a connection-oriented protocol developed primarily for the client-server communication model.

This clause provides some of the key properties and semantics of TCP to better motivate the design decisions in this specification.

### Asymmetric Connection Establishment

TCP recognizes two distinct roles in the communication: “client” and “server.” Connection establishment in TCP is asymmetric because it is always the client who must open the connection to the server, and the server who must accept it.

To establish a connection successfully, the client must know the network address and the port number of the server. The server must be listening on the specified port for connection requests to be detected. Once a connection is established, it can be used for bidirectional data transfer. The connection can be terminated by either side.

It is worth noting that both the “connect” and the “listen” operations are often blocking. When this happens, applications must wait until the connection is established to transmit data. This is in stark contrast with UDP, where applications can send data without waiting for any connection establishment.

### Stream-Oriented Protocol

TCP is stream-oriented, which means that data is delivered to a recipient as a stream of bytes with no user-visible notion of a “message” or a “message boundary.” For applications this implies that the number of read operations at the receiving side is not correlated with the number of write operations on the sender side. Applications cannot assume anything about how TCP packetizes data.

### Reliable Communication

Under normal conditions TCP guarantees reliable, ordered delivery of data. However, TCP’s reliability is limited to a single session (which may be terminated at any point) and only guarantees that data is delivered to the receiver’s socket buffer; not to the middleware internal queues or the application. TCP has no notion of reliability across multiple sessions.

There are at least three failure modes in which TCP cannot guarantee delivery of data: network outage, peer process crash, and peer host crash.

* If a network outage lasts sufficiently long, the sending TCP peer will eventually drop the session losing all the data in its buffer. A subsequent connection has no recollection of the previous connection state.
* In case of a peer process crash, the host operating systems terminates all the active TCP sockets held by the process losing all the data not read by the application.
* Finally, peer host crash is similar to network failure in the sense that the waiting peer eventually drops the session after a retry timeout.

### Head-Of-Line (HOL) Blocking

Applications using TCP may experience a problem where sending independent messages over a TCP connection causes delivery of messages sent later to be delayed within a receiver’s transport layer buffers until an earlier lost message is retransmitted and arrives. HOL blocking is a result of TCP’s order preserving semantics.

### Multicast Support

TCP allows data transmission in strictly unicast fashion. There are exactly two entities participating in TCP connection: a client and a server—no more. If the same data is to be sent to multiple recipients, multiple TCP connections must be established and the data must be copied once for each connection.

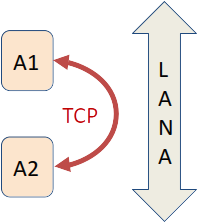
## Deployment Scenarios

This PSM is designed to support a wide variety of deployment scenarios on Local Area Networks (LAN) and Wide Area Networks (WAN).

Two primary considerations are firewalls and Network Address Translation (NAT). Firewalls often restrict incoming and outgoing TCP connections in several ways. For instance, they may disallow all incoming TCP connections but allow outgoing connections. Moreover, on the Internet, physical addresses of hosts are often not visible from outside the organization’s network because of NATs. Considering the wide deployment of both firewalls and NATs, this specification has identified the following deployment categories.

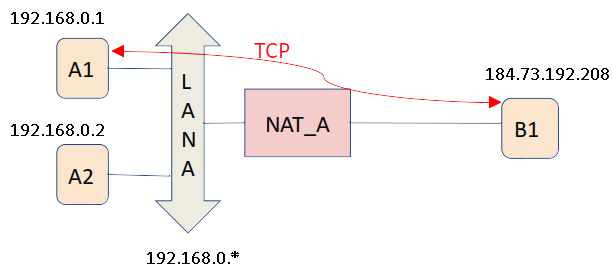
### Direct Communication

In this scenario, depicted in Figure 7.1, two hosts—A1 and A2—can directly connect to each other, and both hosts have their own network address. There is no firewall or NAT.

  
Figure 7.1: Direct Communication

### Client Behind a NAT

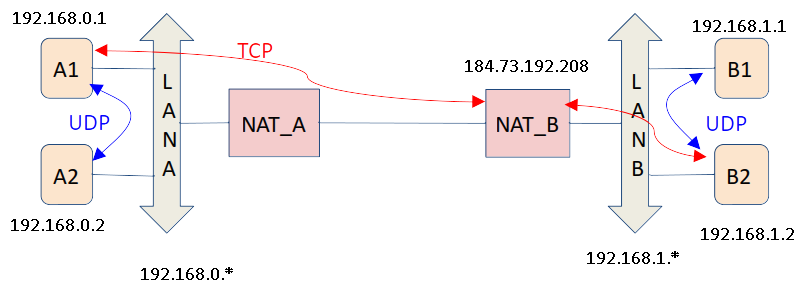
In this scenario, depicted in Figure 7.2, a firewall and NAT are present protecting LAN A. Participants A1 and A2 are behind the NAT. The ports needed to reach A1 and A2 have been opened in the firewall and mapped across the NAT, but with address translation. Participant B1 has a public IP address and can be reached through a well-known port.

  
Figure 7.2: Client Behind a NAT

#### Client and Server Behind NATs

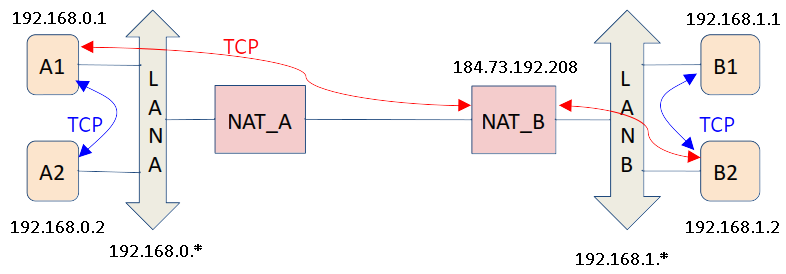
##### Using TCP for WAN Communication Only

This scenario, depicted in Figure 7.3, introduces two different LANs protected by a firewall and NAT. Participants A1 and A2 are behind the NAT\_A, and Participants B1 and B2 are behind NAT\_B. The ports needed to reach A1 and A2 on one side, and B1 and B2 on the other side, have been opened in the corresponding firewalls and mapped across the NATs with address translation. Participants communicate with participants in different NATs using TCP (e.g., A1 with B2), whereas participants within the same LAN communicate over UDP.

  
Figure 7.3: Client and Server Behind NATs—TCP for WAN only

##### Using TCP for both LAN and WAN Communication

This scenario, depicted in Figure 7.4, introduces two different LANs protected by a firewall and NAT. Participants A1 and A2 are behind the NAT\_A, and Participants B1 and B2 are behind NAT\_B. The ports needed to reach A1 and A2 and B1 and B2 been opened in the firewalls and mapped across the NATs with address translation. Participants communicate with other participants over TCP, whether they are in the same LAN or not.

  
Figure 7.4: WAN Deployment Behind NATs—TCP for LAN and WAN

## Technical Issues in using TCP/IP for an RTPS PSM

# DDS-RTPS Platform Specific Model (PSM) for TCP/IP

## Introduction

This chapter defines the Platform Specific Model (PSM) that maps the protocol PIM defined in [DDS-RTPS] to TCP/IP. The goal of this PSM is to define a simple mapping with minimal overhead directly on top of TCP/IP.

## Notational Conventions

This specification uses the same notational as those defined in the definition of the UDP/IP PSM in clause 9.2 of [DDS-RTPS]]. In particular:

* It defines all data types under the RTPS namespace.
* It uses OMG IDL [IDL] for definition of types. (This is a convention for this document. Conformant implementations may support other type representations such as those defined in [DDS-XTYPES].)
* It uses CDR for wire representation. (This is a convention for this document. Conformant implementations may support other type representations such as those defined in [DDS-XTYPES].)

## Mapping of the RTPS Types

The mapping of RTPS types is the same as defined in clause 9.3 in the UDP/IP PSM, except those noted in Table 8.1.

Table 8.1: PSM mapping of the value types that appear on the wire

| Type | Description of the PSM Mapping |
| --- | --- |
| Locator\_t | Mapping of Locator\_t is the same as the mapping defined by the UDP/IP PSM in sub clause 9.3.2 of [DDS-RTPS]. That is:  struct Locator\_t {  long locatorKind;  unsigned long port;  octet[16] address;  };  This specification adds the following Locator\_t kinds to the ones specified by the UDP/IP:  #define LOCATOR\_KIND\_TCPv4 0x04  #define LOCATOR\_KIND\_TCPv6 0x08  Moreover, this specification breaks the 4 bytes assigned to the port field in two different 2-byte groups:   * The first 2 bytes contain the TCPv4/TCPv6 port for server locators. These shall be set to zero for client locators. * The second 2 bytes contain the RTPS logical port. These bytes shall be set to zero when used in control messages affecting the whole connection.   If locatorKind is LOCATOR\_KIND\_TCPv4 the mapping of the 16 bytes of the address field is the following:   * The first 8 bytes correspond to the LAN ID, which uniquely identifies the LAN the locator belongs to. * The following 4 bytes correspond to the WAN IPv4 address; if unavailable it is set to zero. * The last 4 bytes correspond to the LAN IPv4 address.   If locatorKind is LOCATOR\_KIND\_TCPv6, the address field contains an IPv6 address. IPv6 addresses typically use a shorthand hexadecimal notation that maps one-to-one to the 16 octets in the address field.  For example the representation of the IPv6 address FF00:4501:0:0:0:0:0:32 is:  address = (0xff,0,0x45,0x01,0,0,0,0,0,0,0,0,0,0,0,0x32} |

## Mapping of RTPS Messages

### Overall Structure

The RTPS PIM defines the overall structure of a message, which is composed of a header and a set of submessages.

A key difference between the UDP and TCP is that UDP has a user visible notion of message that TCP lacks due to its stream-oriented nature. Therefore, this PSM shall define a way to specify the length of every RTPS message to provide an appropriate encapsulation.

To simplify the implementation, allowing the same engine to parse RTPS messages over TCP and UDP, this PSM defines a new RTPS TCP/IP PSM header that shall prepend every RTPS message—as defined in the UDP/IP PSM in clause 9.4 of [DDS-RTPS]—over the TCP/IP transport. The role of this new header is to provide all the information about the message length and message integrity that is not part of the RTPS header.

The resulting layout of an RTPS over TCP message is the following:

Message:

0...2...........7...............15.............23...............31

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| RTPS TCP/IP PSM Header (RTCP Header) |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| {UDP/IP RTPS Message OR RTPS TCP/IP PSM Control Message} |

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

As described above, besides UDP/IP RTPS messages, the RTPS TCP header may also be followed by RTPS TCP Control Protocol Messages (which are defined in chapter 9).

Both UDP/IP RTPS messages and RTPS TCP Control Protocol shall be aligned at a 32-bit boundary with respect to the start of the message.

### Mapping of PIM SubmessageElements

This PSM preserves the IDL type and on-the-wire representation defined by the RTPS UDP/IP PSM in sub clause 9.4.2 of [DDS-RTPS] for all SubmessageElements except for LocatorList, which is represented as defined below.

#### LocatorList

The PSM mapping for the LocatorList SubmessageElement is the same as that defined in sub clause 9.4.2.10 of [DDS-RTPS]. However, the wire representation of Locator\_t changes as a result of the mapping of defined by this specification in Table 8.1.

For a Locator\_t with LOCATOR\_KIND\_TCPv4, the wire representation is the following:

Locator\_t:

0...2...........8...............16.............24...............31

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| locatorKind = LOCATOR\_KIND\_TCPv4 |

+---------------+---------------+---------------+---------------+

| tcpPort (2 bytes) | logicalPort (2 bytes) |

+---------------+---------------+---------------+---------------+

| |

~ Unique LAN ID (8 bytes) ~

| |

+---------------+---------------+---------------+---------------+

| WAN IPv4 Address (4 bytes) |

+---------------+---------------+---------------+---------------+

| LAN IPv4 Address (4 bytes) |

+---------------+---------------+---------------+---------------+

For a Locator\_t with LOCATOR\_KIND\_TCPv6, the wire representation is the following:

Locator\_t:

0...2...........8...............16.............24...............31

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| locatorKind = LOCATOR\_KIND\_TCPv6 |

+---------------+---------------+---------------+---------------+

| tcpPort (2 bytes) | logicalPort (2 bytes) |

+---------------+---------------+---------------+---------------+

| |

~ ~

| |

~ IPv6 address (16 bytes) ~

| |

~ ~

| |

+---------------+---------------+---------------+---------------+

### Additional SubmessageElements

This specification does not introduce any additional submessage elements to those defined by the RTPS UDP/IP PSM in sub clause 9.4.3 of [DDS-RTPS].

### Mapping of the RTPS Header

Sub clause 8.3.3 of [DDS-RTPS] specifies that all RTPS messages shall include a leading RTPS header. This PSM preserves the mapping of the RTPS Header defined by the RTPS UDP/IP PSM in sub clause 9.4.4 of the same specification.

As mentioned above, this PSM adds a new RTPS TCP/IP PSM header—hereinafter named RTCP header—that indicates the length of the subsequent RTPS or RTPS TCP Control Protocol message.

The layout of the RTCP Header is the following:

Header:

0...2...........8...............16.............24...............32

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

| 'R' | 'T' | 'C' | 'P' |

+---------------+---------------+---------------+---------------+

+ length +

+---------------+---------------+---------------+---------------+

+ CRC +

+---------------+---------------+---------------+---------------+

+ logicalPort |

+---------------+---------------+

Where:

* The RTPS TCP/IP PSM Protocol Identifier (4 bytes) contains the characters "R", "T", "C", and "P". It is used for checking message synchronization.
* Message length (4 bytes) indicates the length of the whole RTPS TCP message, including the header.
* CRC (4 bytes) is used for checking message integrity.
* Logical port (2 bytes), where the port 0x00 0x00 is reserved to identify TCP/IP PSM Control Messages.

### Mapping of the RTPS Submessages

The mapping of RTPS submessages remains the same as defined in the RTPS UDP/IP PSM, sub clause 9.4.5 of [DDS-RTPS].

## RTPS Message Encapsulation

The RTCP header encapsulates RTPS messages and RTCP Control Protocol messages providing the length that delimits them within the TCP data stream.

## Mapping of the RTPS Protocol

### Default Locators

TCP does not support multicast and therefore, this PSM supports only unicast locators for both discovery and user traffic.

This specification uses the same formulas and default values for computing port numbers that the UDP/IP PSM specifies in sub clause 9.6.1 of [DDS-RTPS]. However, a key difference between the RTPS UDP/IP PSM and the RTPS TCP/IP PSM is the interpretation and use of these port numbers. In UDP, the computed port numbers correspond to physical ports where discovery and user traffic are sent. In contrast, the RTPS TCP/IP PSM interprets the computed port numbers as logical ports that classify the discovery and user traffic, and almost never corresponds to the actual port the TCP connection is established through.

This specification does not support announcing or listening on a multicast address. The default announcement rate is the same as the rate defined by the UDP/IP PSM in sub clause 9.6.1.4.2 of [DDS-RTPS].

### Data Representation for the Built-in Endpoints

The IDL and wire representation of the built-in endpoints are the same as defined in sub clause 9.6.2 of the UDP/IP PSM [DDS-RTPS].

### ParameterId Definitions Used to Represent In-line QoS

# RTPS TCP Control Protocol (TO BE DISCUSSED -- needs agreement)

## Overview

To bootstrap communication between clients and servers, this specification defines a simple control protocol that ensures that different kinds of RTPS messages can be exchanged over one or multiple TCP connections.

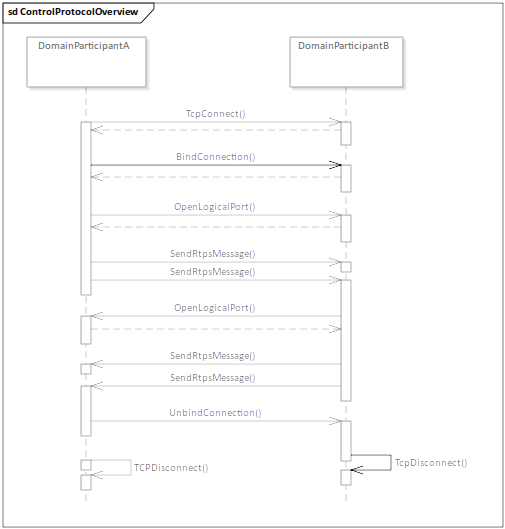
The capabilities of this control protocol include:

1. Identifying a connection as an RTCP connection (BindConnection).
2. Requesting a destination port (OpenLogicalPort and LogicalPortIsClosed).
3. Checking whether a range of ports are available (CheckLogicalPorts).
4. Checking whether a connection is still alive (KeepAlive).
5. Finalizing an RTCP session (UnbindConnection).

## Control Protocol Behavior

The sequence diagram in Figure 9.1, along with a short description of the interactions, provides a general behavioral overview of the Control Protocol.

1. A client application (i.e., a DDS DomainParticipant) initiates communication by opening a connection to a well-known TCP port in which a server (i.e., another DDS DomainParticipant) is listening (e.g., 7400), and the server accepts the connection (i.e., TCP's 3-way handshake succeeds).
2. The server is unaware of the intent behind the connection opened in step 1; therefore, the client must send an identity indication using a BindConnection request message. Then, the server accepts the BindConnection request and sends a BindConnection response.
3. The client sends an OpenLogicalPort request to be able to send RTPS TCP messages to that specific logical port. The server accepts the OpenLogicalPort request.
4. The client can now send RTPS messages to the server using the mappings defined in clause 8.4.
5. The server issues an OpenLogicalPort request to open a logical port that allows it to send RTPS messages. The client accepts the OpenLogicalPort request sending an OpenLogicalPort response.
6. The server can now send RTPS messages to the client using the mappings defined in clause 8.4.
7. The client issues an UnbindConnection request indicating its willingness to close the TCP connection and closes the TCP connection.
8. Upon the reception of the UnbindConnection message, the server closes the TCP connection as well.

  
Figure 9.1: Control Protocol Overview

## Control Protocol Messages

Control protocol messages shall be preceded by the RTPS TCP/IP header defined by this PSM in sub clause 8.4.4. Each control protocol message has a control message header—which marks the beginning of a control message—and a control message data component.

### Control Protocol Message Header

This header includes a CtrlPrtlMsgKind that identifies the kind of message, followed by a set of flags that indicates certain features of the message, a message length, and a TransactionId that uniquely identifies a control message request.

The wire representation of the control protocol message header is described below.

Header:  
0...2...........8...............16.............24...............31

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

|CtrlPrtlMsgKind| flags |R|P|E| length |

+---------------+---------------+---------------+---------------+

| |

~ ~

| TransactionId (12 bytes) |

~ ~

| |

+---------------+---------------+---------------+---------------+

#### Control Protocol Message Kind

As previously stated, this specification defines are six control protocol messages: BindConnection, UnbindConnection, OpenLogicalPort, CheckLogicalPorts, LogicalPortIsClosed, and KeepAlive. These control protocol messages are exchanged in a request/response fashion, and are uniquely identified by their 8-bit wide CtrlPrtlMsgKind.

Table 9.1 specifies the value assigned to each control protocol message.

Table 9.1: Control message kinds and their 8-bit corresponding value

|  |  |
| --- | --- |
| Control Protocol Message Kind | Value |
| BindConnection Request | 0x0D1 |
| BindConnection Response | 0x0E1 |
| OpenLogicalPort Request | 0x0D2 |
| OpenLogicalPort Response | 0x0E2 |
| CheckLogicalPorts Request | 0x0D3 |
| CheckLogicalPorts Response | 0x0E3 |
| KeepAlive Request | 0x0D4 |
| KeepAlive Response | 0x0E4 |
| LogicalPortIsClosed Request | 0x0D5 |
| UnbindConnection Request | 0x0D6 |

#### Flags

In addition, the control protocol message header contains of a set of flags to: (1) identify the endianness used to encode the submessage, (2) indicate whether the submessage contains a payload, and (3) indicate whether the submessage requires a response.

* The EndiannessFlag is the least-significant bit of the flags. The EndiannessFlag is represented with the literal "E". E=0 means big-endian, E=1 means little-endian. The value of the EndiannessFlag can be obtained from the expression:  
  E = ControlMessageHeader.flags & 0x01
* The HasPayloadFlag is the second least-significant bit of the flags. The HasPayloadFlag is represented with the literal "P". P=0 means that the submessage does not contain a payload, P=1 means that the submessage contains a payload. The value of the HasPayloadFlag can be obtained from the expression:  
  P = ControlMessageHeader.flags & 0x02
* The RequiresResponseFlag is the third least-significant bit of the flags. The RequiresResponseFlag is represented with the literal "R". R=0 means that the submessage does not require a response, R=1 means that the submessage requires a response. The value of the RequiresResponseFlag can be obtained from the expression:  
  R = ControlMessageHeader.flags & 0x03

#### Length

The length is the number of octets from the first octet of the contents of the control protocol message until the first octet of the header of the next control protocol or RTPS message. The representation of this field is a CDR unsigned short (2 bytes).

#### Transaction ID

All messages contain a 12-byte TransactionId that correlates a request with a response.

### Control Message Data

#### Control Protocol Request Data

The control message follows the Control Message header defined in sub clause 9.3.1. It shall be serialized following the CDR serialization the ControlProtocolRequest union according to the rules defined in [DDS-XTYPES]. The RequestKind union discriminator specifies the kind of message that gets serialized.

struct ConnectionRequest\_t {

ProtocolVersion\_t protocolVersion;

VendorId\_t vendorId;

Locator\_t transportLocator;

};

struct OpenLogicalPortRequest\_t {

unsigned short logicalPort;

};

struct CheckLogicalPortsRequest\_t {

sequence<unsigned short> logicalPortsRange;

};

struct KeepAliveRequest\_t {

Locator\_t locator

};

struct LogicalPortIsClosedRequest\_t {

unsigned short logicalPort;

};

enum RequestKind {

BIND\_CONNECTION,

OPEN\_LOGICAL\_PORT,

CHECK\_LOGICAL\_PORT,

KEEP\_ALIVE

UNBIND\_CONNECTION,

LOGICAL\_PORT\_IS\_CLOSED

};

union RequestData switch(RequestKind requestKind) {

case BIND\_CONNECTION:

ConnectionRequest\_t connectionRequest;

case OPEN\_LOGICAL\_PORT:

OpenLogicalPortRequest\_t openLogicalPortRequest;

case CHECK\_LOGICAL\_PORT:

CheckLogicalPortsRequest\_t checkLogicalPortsRequest;

case KEEP\_ALIVE:

KeepAliveRequest\_t keepAliveRequest;

case LOGICAL\_PORT\_IS\_CLOSED:

LogicalPortIsClosedRequest\_t logicalPortIsClosedRequest;

case UNBIND\_CONNECTION:

// empty

};

struct ControlProtocolRequestData {

RequestData requestData;

};

##### BindConnection Request

The purpose of the BindConnection request is to provide some information for the context of the connection that the client has established with the server. This includes information about the protocol version, the vendor implementation, and the locator to reach the client application. Therefore, this type of request can only be sent from a client to a server and shall be the first message it sends after establishing a connection.

BindConnection requests shall set both the HasPayloadFlag and RequiresResponseFlag to 1, and set the EndiannessFlag with the right endianness.

ConnectionRequest\_t shall be set with the following attributes:

* ProtocolVersion\_t protocolVersion, version of the RTPS TCP/IP PSM standard.
* VendorId\_t vendorId, data structure identifying the vendor implementing the RTPS TCP/IP PSM.
* Locator\_t clientLocator, locator to reach the client application.

##### OpenLogicalPort Request

Both clients and servers can send an OpenLogicalPort requests to announce their intent to send data using the requested logical port. Therefore, this kind of request must precede any RTPS TCP message directed to that logical port over the TCP connection.

OpenLogicalPort requests shall set both the HasPayloadFlag and RequiresResponseFlag to 1, and set the EndiannessFlag with the right endianness.

OpenLogicalPortRequest\_t shall be set with the following attribute:

* unsigned short logicalPort, the requested logical port.

##### CheckLogicalPorts Request

Both clients and servers can send CheckLogicalPorts requests to figure out whether a range of logical ports are available on the other end.

CheckLogicalPortsRequest\_t shall be set with the following attribute:

* sequence<unsigned short> logicalPortsRange, range of requested logical ports.

##### Keep Alive Request

Both clients and servers can send KeepAlive requests to check whether the connection is alive. If the a response does not arrive before a timeout, the connection is considered dead and the sender shall close the connection following the steps described in the UnbindConnection request.

KeepAlive requests shall set both the HasPayloadFlag and RequiresResponseFlag to 1, and set the EndiannessFlag with the right endianness.

KeepAliveRequest\_t shall be set with the following attribute:

* Locator\_t locator, locator to reach the client or server application issuing the KeepAlive request.

##### LogicalPortIsClosedRequest

Both clients and servers can send LogicalPortIsClosed requests to indicate that a logical port is closed or that the other end has incorrectly sent an RTPS TCP message to a port that is closed. Upon the reception of this message, the receiver must stop sending RTPS TCP messages to that logical port.

KeepAlive requests shall set the HasPayloadFlag to 1, the RequiresResponseFlag to 0, and set the EndiannessFlag with the right endianness. Therefore, this kind of request shall not be followed by a response upon its reception.

LogicalPortIsClosedRequest\_t shall be set with the following attribute:

* unsigned short logicalPort, the logical port that is marked as closed.

##### UnbindConnection Request

Both clients and servers can send an UnbindConnection request to indicate the intent to close the current TCP connection. After sending this message, the sender shall close the TCP connection. Likewise, the receiver of this request shall close the TCP connection upon the reception of the request.

UnbindConnection requests shall set the HasPayloadFlag to 0, the RequiresResponseFlag to 0, and set the EndiannessFlag with the right endianness. Therefore, this kind of request shall not be followed by a response upon its reception and shall not set any attribute in the ControlProtocolRequestData.

#### Control Protocol Response Data

The control protocol response data follows the Control Message header defined in sub clause 9.3.1. The resulting wire representation shall be the result of applying the CDR serialization rules defined in [DDS-XTYPES] for the ControlProtocolRequestData structure, which includes a ResponseCode indicating the success or failure of the action triggered by the request message, and a ResponseData that depends on the ResponseKind union discriminator.

enum ResponseCode {

RETCODE\_OK,

RETCODE\_BAD\_REQUEST,

RETCODE\_EXISTING\_CONNECTION,

RETCODE\_INVALID\_PORT,

RETCODE\_UNKNOWN\_LOCATOR,

RETCODE\_INCOMPATIBLE\_VERSION,

RETCODE\_SERVER\_ERROR

};

enum ResponseKind {

BIND\_CONNECTION,

OPEN\_LOGICAL\_PORT,

CHECK\_LOGICAL\_PORT,

KEEP\_ALIVE

};

struct BindConnectionResponse\_t {

Locator\_t locator;

};

struct CheckLogicalPortsResponse\_t {

sequence<unsigned short> availableLogicalPorts;

};

union ResponseData switch(ResponseKind response\_kind){

case BIND\_CONNECTION:

BindConnectionResponse\_t bindConnectionResponse;

case OPEN\_LOGICAL\_PORT:

case CHECK\_LOGICAL\_PORT:

case KEEP\_ALIVE:

// empty

};

struct ControlProtocolResponseData {

ResponseCode responseCode;

ResponseData responseData;

};

##### BindConnection Response

Upon the reception of a BindConnection request, servers shall issue a BindConnection response to indicate whether they accept the use of the TCP connection for the binding parameters.

BindConnection responses shall set the HasPayloadFlag to 1, the RequiresResponseFlag to 0, and the EndiannessFlag with the right endianness.

In case of an error, the server shall set ResponseCode to one of the following error messages:

* RETCODE\_EXISTING\_CONNECTION, in case the client has performed a BindConnection request as part of the current TCP connection.
* RETCODE\_INCOMPATIBLE\_VERSION, if the version of the RTPS TCP/IP PSM protocol the client is using is incompatible with the version the server is using. In that case, the server shall close the TCP connection after sending the response.
* RETCODE\_BAD\_REQUEST, in case of a malformed request.
* RETCODE\_SERVER\_ERROR, in case of any other kind of server error.

If the server accepts the connection, then it shall set the ResponseCode to RETCODE\_OK.

Regardless of the success or failure of the operation, the server shall set BindConnectionResponse\_t with the following attributes:

* Locator\_t serverLocator, locator to reach the server application.

##### OpenLogicalPort Response

Clients or servers receiving an OpenLogicalPort request shall issue a response indicating whether they accept or reject the use of the requested logical port.

OpenLogicalPort responses shall set the HasPayloadFlag to 1, the RequiresResponseFlag to 0, and the EndiannessFlag with the right endianness.

In case of an error, the receiver of the OpenLogicalPort request shall set ResponseCode to one of the following error messages:

* RETCODE\_INVALID\_PORT, in case the requested port is already in use. Requests RTPS TCP sent to an invalid port shall be dropped.
* RETCODE\_BAD\_REQUEST, in case of a malformed request.
* RETCODE\_SERVER\_ERROR, in case of any other kind of server error.

If the receiver of the OpenLogicalPort request agrees on using the requested port, it shall set ResponseCode to RETCODE\_OK.

##### CheckLogicalPortsResponse

Clients or servers receiving a CheckLogicalPorts request shall issue a response indicating which of requested logical ports are available.

CheckLogicalPorts responses shall set the HasPayloadFlag to 1, the RequiresResponseFlag to 0, and the EndiannessFlag with the right endianness.

In case of an error, the receiver of the CheckLogicalPorts request shall set ResponseCode to one of the following error messages:

* RETCODE\_BAD\_REQUEST, in case of a malformed request.
* RETCODE\_SERVER\_ERROR, in case of any other kind of server error.

If the received request is valid, regardless of whether the requested ports are available or not, the receiver of the CheckLogicalPorts request shall set ResponseCode to RETCODE\_OK and the availableLogicalPorts attribute of BindConnectionResponse\_t as follows:

* sequence<unsigned short> availableLogicalPorts, shall be populated with all the available ports that were requested. If none of the requested ports is available, the availableLogicalPorts sequence shall be empty.

##### KeepAlive Response

Clients or servers receiving a KeepAlive request shall issue a KeepAlive response to assert that the connection is still alive.

In case of an error, the server shall set ResponseCode to one of the following error messages:

* RETCODE\_UNKNOWN\_LOCATOR, if the sender of the response does recognize the sender of the request. As a result of that, both sides shall close the connection.
* RETCODE\_BAD\_REQUEST, in case of a malformed request.
* RETCODE\_SERVER\_ERROR, in case of any other kind of server error.

If the receiver of the KeepAlive recognizes the locator of the application sending the request, it shall set ResponseCode to RETCODE\_OK to acknowledge that the connection is alive.

# Use of the RTPS TCP/IP PSM (non-normative)

This chapter describes the use of the RTPS PSM defined in chpater 8 and the RTPS TCP Control Protocol defined in chpater 9 to address implementing the different scenarios described in clause 7.4.

The goal is to provide implementers of this specification with enough information on how the different building blocks defined address the challenges proposed by real-world scenarios.

## Overview

Let us assume an implementation of DDS that uses DDS-RTPS for interoperable communications. This implementation is based on a pluggable transport architecture that allows RTPS to function over different transport protocols such as UDP. The adoption of this specification results therefore in the implementation of a new TCP transport plugin.

Transport plugins are unaware of the intent of the upper layers that instruct them to deliver a set of messages. They work solely with a set of locators that identify where information needs to be delivered. Upper layers, more specifically the ones that controlling the functionality of DDS DomainParticipants, are responsible for invoking the appropriate methods of the transport plugin API to instruct it to deliver messages to the appropriate destination.

## Use of RTPS TCP PSM/IP PSM in Deployment Scenarios

### LAN Deployment

The LAN deployment scenario depicted in Figure 7.1 (sub clause 7.4.1) presents two DomainParticipants A1 and A2, which belong to the same LAN and wish to communicate with one another.

As described in Figure 10.1 (which depicts in detail the use of the RTPS TCP/IP PSM for this scenario), A1 registers an instance of the TCP transport plugin that enables it to communicate with other DomainParticipants over TCP. A1 has been configured with a set of locators to it needs to contact to announce its presence via SPDPdiscoveredParticipantData messages. The list of locators includes the IP address and physical port to reach A2, which has also registered an instance of the TCP transport plugin to communicate with other DomainParticipants via TCP. It important to note that in this example, A2 does not have A1 in its initial list of locators.

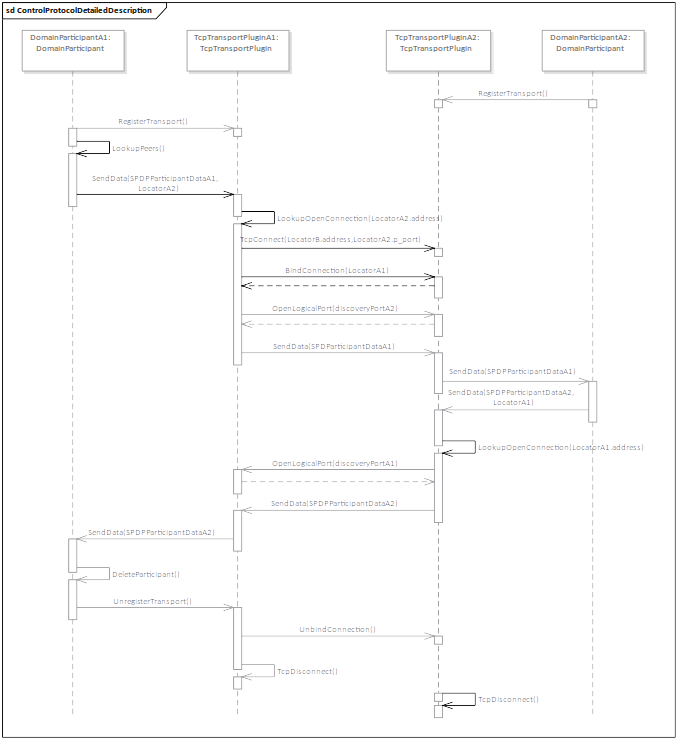
After looking up the list of peers it needs to contact, A1 asks the TCP transport plugin to deliver a SPDPdiscoveredParticipantData message to A2 (identified by its locator). Next, the TCP transport checks the list of existing TCP connections with the locator that identifies A2, and after not finding any existing connection, establishes a new connection with A2’s TCP transport using the A2’s IP address and physical port that in A2’s locator.

Once the TCP connection has been established, A1 identifies the connection by sending BindConnection request. This operation provides the protocol version, vendor, and Locator through which A1 can be reached. This information must be saved by the TCP transport plugin to identify the connection so that it can be reused for subsequent messages to A1.

Next, the transport plugin announces its intent to send data to discoveryPortA2, which is the appropriate logical port in A2 to which the SPDPdiscoveredParticipantData message needs to be sent, using the OpenLogicalPortOperation. The successful response authorizes the transport to deliver the message to A2’s transport plugin, which then forwards the SPDPdiscoveredParticipantData to A2.

At this point A2 has discovered A1 and adds it to the list of remote DomainParticipants. After a period of time A2 sends its SPDPdiscoveredParticipantData to A1. The delivery process is similar to the one we have just described. The main difference is that when A2 asks the TCP transport plugin to deliver the SPDPdiscoveredParticipantData to A1, A2’s TCP transport plugin realizes that there is an existing connection between A1 and A2 that can be reused. Therefore, because the connection is established and bound, A2’s TCP transport plugin only needs to request access to discoveryPortA1 sending an OpenLogicalPort request. Once permission is granted, the TCP transport plugin delivers the message to A1’s TCP plugin, which delivers it to A1.

Finally, the user deletes DomainParticipant A1, which triggers a process to unregister the TCP transport plugin, which sends a UnbindConnection request to A2’s TCP transport plugin, after which both transport plugins close the TCP connection.

  
Figure 10.1: LAN Deployment Detailed Example

### WAN Deployments