Master's Thesis

Czech Technical University in Prague

F3

**Faculty of Electrical Engineering
Department of measurement**

Implementation of actual version of DDSI-RTPS protocol for distributed control in Ethernet network

Jiri Hubacek
Open Informatics

January 2016
Supervisor: Pavel Pisa

Acknowledgement / Declaration

Podekovani..

Prohlasuji..

. . . .

iii

Draft: 12. 12. 2015

Abstrakt / Abstract

Czech abstract..

Klíčová slova: RTPS, ORTE, Ethernet, Real-Time

Překlad titulu: Implementace aktuální verze protokolu DDSI-RTPS pro distribuované řízení v síti Ethernet

English abstract..

Keywords: RTPS, ORTE, Ethernet, Real-Time

/ **Contents**

Chapter 1
Introduction

The Real-Time Publish-Subscribe (RTPS)[1] is the protocol of Data Distribution Service (DDS)[2] family, supporting Data-Centric Publish-Subscribe in real time and specifying communication in a decentralized network, where multiple nodes needs to send and/or receive data in real time. Specification of protocol is being developed by Object Management Group[3] - international, open membership, not-for-profit technology standards consortium, since version 1.0 on February 2002 till version 2.2 on September 2014.

This thesis aims on upgrading ORTE implementation of RTPS protocol to be compatible with the latest standard version 2.2. The structure is as follows. In Chapter 1, there is an introduction to RTPS and ORTE. Chapter 2 compares implemented RTPS 1.0 with the latest RTPS 2.2, chapter 3 covers changes needed for compatibility with version 2.2 of the RTPS protocol and chapter 4 covers testing of new implementation of RTPS protocol in ORTE. In chapter 5, demo application of ORTE called *Shape* for Android is introduced. The application was developed as part of familiarization with ORTE, therefore ORTE with RTPS 1.0 implementation is used in this application. Security for DDS is discussed in chapter 6.

1.1 DDS

There are two main models used in Data Distribution Services. *Centralized* model, where single server for the whole network is needed and all communication goes throw it, introduces single point of failure. When the server is unreachable, the whole network is non-functional. By contrast, *decentralized* approach has no central server, no single point of failure. When one node of the network is non-functional, the rest of the network can continue in data transfers.

1.2 DCPS

In the Data-Centric Publish-Subscribe network, data are sent by *Publishers* and received by *Subscribers*. Node can be *Publisher*, *Subscriber* or both and each node can be interested in different data, timing and reliability. Data-Centric Publish-Subscribe network is responsible for delivery of right data between right nodes with right parameters.

1.3 RTPS

Real-Time Publish-Subscribe is wire protocol developed to ensure interoperability between DDS implementations. It has been designed to be fault tolerant (decentralized), scalable, tunable, with plug-and-play connectivity and ability of best-effort and reliable communication in real time applications.

▉ 1.4 ORTE

Open Real-Time Ethernet (ORTE)[4] is the implementation of RTPS 1.0. It's implemented in Application layer of UDP/IP stack, written in C, under open source license, with own API. Because there are no special requirements, it should be easy to port ORTE to many platforms, where UDP/IP stack is implemented.

▉ 1.5 Symbols

CORBA	Common Object Request Broker Architecture
DCPS	Data-Centric Publish-Subscribe
DDS	Data Distribution Service
HMAC	Hash-based Message Authentication Code
IP	Internet Protocol
IPsec	IP Security
MAC	Message Authentication Code
OMG	Object Management Group
ORTE	Open Real-Time Ethernet
RTPS	Real-Time Publish-Subscribe
SEDP	Simple Endpoint Discovery Protocol
SPDP	Simple Participant Discovery Protocol
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol

Chapter 2
Versions comparation

Chapter 3
Compatibility changes

Chapter 4
Testing of implementation

Chapter 5
Shape for Android

5.1 Shape demo

With ORTE implementation of RTPS 1.0 protocol, demo application called Shape is delivered. Shape demo demonstrates the functionality of ORTE - when the color (Blue, Green, Red, Black, Yellow) is choosen, the *Publisher* is created as random shape (Circle, Square, Triangle) moving on the screen. Then, under the topic of color name, object's shape, color and coordinates are published to the network. It's possible to receive and interpret object's data (to see colored shapes moving on the screen) by adding the *Subscribers* of specific topics (colors).
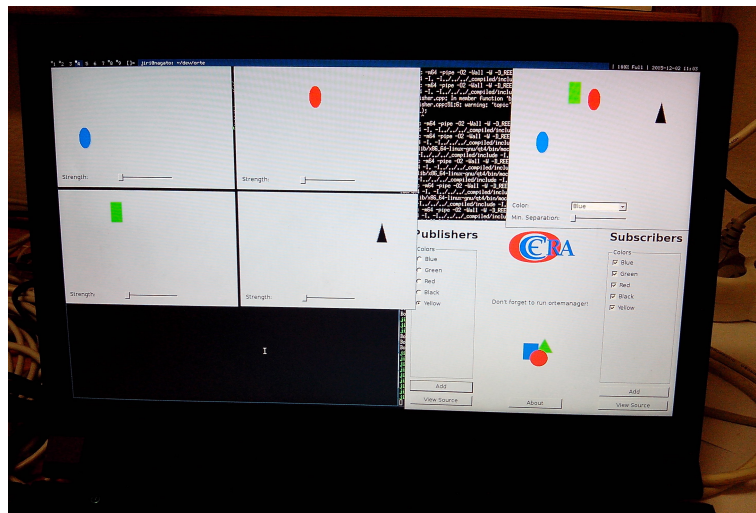

Figure 5.1. Shape demo - *Publishers* and *Subscribers*

5.2 Familiarization with ORTE

The familiarization with ORTE was done by creating demo application for Android compatible with Shape. Because the port of ORTE to Android has been already done in [5] and is available as library, the main task was application design and compatibility ensurance. The application was designed to be as simple as possible. *Publishers* view allows to create new *Publisher* of specific color and random shape, *Subscribers* view allows to set up *Subscribers* of specified colors. Finally, Settings and Help views are present.

5.3 Classes

As in Shape demo, in Shape for Android the *BoxType* class is presented, allowing to create, send and receive objects. *BoxType* consists of *color* (integer), *shape* (integer)

and *rectangle* (*BoxRect*), where *BoxRect* is class for storing coordinates - *top_left_x* (short), *top_left_y* (short), *bottom_right_x* (short), *bottom_right_y* (short). The *BoxType* is extension of *MessageData* class delivered with ORTE library for Android. It allows to send and receive objects.

PublisherShape class stores *BoxType* information about *Publisher*, it's properties needed for ORTE, methods for communication with object and prepares data to send. In *Publisher* view, *Publisher* objects are created, stored in *ArrayList* and drawed on screen. Data objects are sent in *Publisher* activity each time objects are redrawn.

SubscriberElement class receives *BoxType* object from ORTE and stores it's data and methods needed for presentation. In *Subscriber* view, all received objects are stored in *ArrayList* and periodically redrawn.

Settings view allows to set up scaling, needed because of various dimensions of screens. It also contains a list of managers - in RTPS 1.0 special application called manager is used for communication of available *Publishers/Subscribers* between nodes. In RTPS 2.2 Simple Participant Discovery Protocol (SPDP) and Simple Endpoint Discovery Protocol (SEDP) are used.

Help view contains information about ORTE, Shape and application usage.

5.4 Compatibility

BoxType in Shape and Shape for Android is a little bit different. The reason is just familiarization with ORTE implementation and RTPS protocol, where misunderstooding was not fully avoided. Suggestions for improvements follows.

The first property of *BoxType* is *color*. In Shape demo, *color* is typed as CORBA_octet (macro for uint8_t, 1 byte) and in Shape for Android, *color* is of integer type (4 bytes). The reason why this approach does not break the compatibility is following: each data-type serialized by CORBA is aligned to 4 byte boundary. In this case, object color is first byte and the rest until the boundary is filled by zero bytes. This data representation corresponds to Little Endian in which the message is encoded by default (endianness is operating system dependent), so when Shape for Android deserialize data, Little Endian encoded integer is obtained. It also works in opposite direction - value of the *color* is serialized as integer, encoded as Little Endian and on the side of Shape demo, CORBA_octet is deserialized and 3 zero bytes skipped because of boundary alignment. The problem could arise when *color* would be sent as integer with Big Endian encoding and received as CORBA_octet, because the value of the first byte would be then zero. Also, the problem wouldn't persist in the opposite direction, because endianness is always part of the RTPS message so even node with Big Endian default encoding would receive Little Endian encoded message correctly.

The second property of *BoxType* is *shape*. The type in Shape demo is CORBA_long (macro for int32_t, 4 bytes) and integer (4 bytes) in Shape for Android. Therefore there is no problem with *shape* property.

The last property of *BoxType* is *BoxRect* consisting of coordinates of object. Each value of *BoxRect* is CORBA_short type (2 bytes) in Shape demo and short type (2 bytes) in Shape for Android. Because *BoxRect* is presented as CORBA autonomous data-type, the whole data-type (8 bytes) is aligned to 4 bytes boundary.

The suggestion for the future improvement of Shape demo and Shape for Android is the revision of *BoxType* data-type.
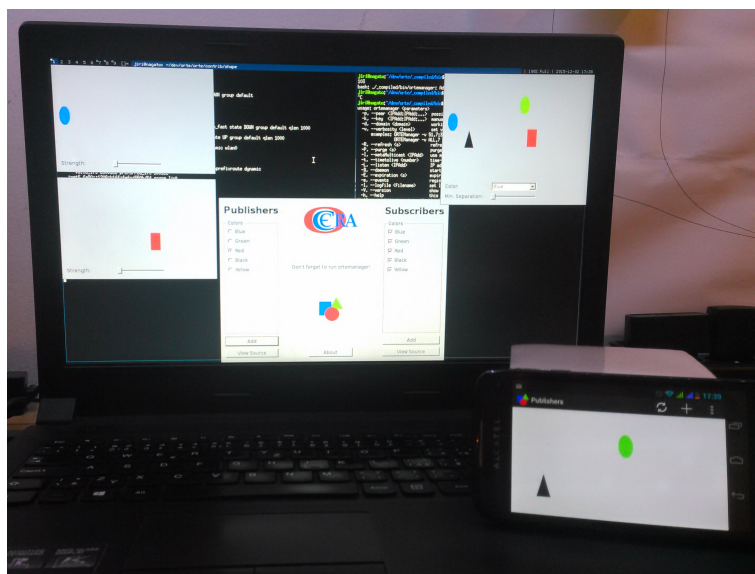
Figure 5.2. Shape for Android - Publishers view

Chapter 6
Security for DDS

In the modern world, security is often considered. Technologies for securing communication differs by TCP/IP layers [6] - security at Media access layer consists of preventing deterioration of physical media, environmental noise and access to media. At Network layer, IPsec (IP Security Architecture) protocol is used while Transport layer uses TLS (Transport Layer Security) protocol. In this chapter, Application layer security for DDS standard [7] and possibilities of implementation in RTPS protocol are considered.

6.1 Threats

From point of view of DDS standard, communication takes place in the domain consisting of participants with various number of publishers and subscribers. In this context, Application layer security threats are following:

- Unauthorized[1]) subscription
- Unauthorized publication
- Tampering and replay
- Unauthorized access to data

Unauthorized subscription is a situation when malicious participant receives data for which it is not allowed to. In network infrastructure where access to media is shared, communication runs over multicast or participants sits on one node, it's practically unavoidable to restrict access to data. The solution is making data unreadable for malicious participant - in other words, applying encryption on publisher's side and sharing keys with authenticated subscribers only.

When malicious participant attempts to send data which it is not allowed to, it's called *Unauthorized publication*. For subscriber it's important to receive data only from valid publishers to avoid influence of malicious participant on data. The solution is to include authentication information to data sent by valid publishers so subscribers would be able to recognize data by authenticated publishers from data sent by malicious participant. Two ways how to accomplish authentication of publishers in data are Hash-based message authentication code (HMAC) and digital signature. HMAC creates authentication code using secret key shared between publisher and subscriber. Digital signature is based on private/public key pair - authentication code is created as message digest encrypted by private key of publisher. Each subscriber has access to public key of publisher and can use it to decrypt the authentication code to message digest and compare it with message digest calculated by itself. The point is that these two message digests equals if and only if the authentication code is encrypted by publisher's private key and decrypted by publisher's public key. Digital signature is called *asymmetric*

---

[1]) Difference between authentication and authorization has to be clear. Authentication is verification of (in this context) participant - that the participant is really the one it claims to be. On the other hand, authorization is process of allowing access to data for already authenticated participant.

cryptography (private/public key pair) and is much slower then *symmetric cryptography* (shared key), therefore the use of HMAC is preferred because of performance reasons.

Valid publisher would send data to subcsriber and malicious participant (now, malicious participant will be allowed to subscribe but not to publish). However if the same key is shared between publisher, subscriber and malicious paritcipant, there is no way how to prevent malicious participant to use this shared key for mimicking publisher and sending data to subscriber. This threat is called *Tampering and Replay* and can be solved by sharing different keys between publishers and subscribers. When the communication is taken over multicast, multiple HMACs are needed to be included in data, but this solution is still more powerful than using digital signatures.

In the DDS network, some devices acts as relay devices forwarding data. These devices need to be trusted as valid publishers and subscribers, but it's not always desirable to let them understand data they work with. The solution for *Unauthorized Access to Data* is having different keys for HMAC and data encryption and to share keys for decrypting of data only with desired endpoint devices.

6.2 Securing RTPS Messages

References

[OMG:DDSI-RTPS22] [1] Object Management Group (OMG). *The Real-Time Publish-Subscribe Protocol (RTPS) DDS Interoperability Wire Protocol Specification* . 2014 .
http://www.omg.org/spec/DDSI-RTPS/2.2/ .

[OMG:DDS] [2] Object Management Group (OMG). *Data Distribution Service for Real-Time Systems* . 2007 .
http://www.omg.org/spec/DDS/ .

[www:OMG] [3] *Object Management Group* .
http://www.omg.org/index.htm .

[FEE:ORTE] [4] *ORTE - Open Real-Time Ethernet* .
http://orte.sourceforge.net/ .

[FEE:vajnar-b] [5] *ORTE communication middleware for Android OS* . 2014 .

[www:tcp-ip] [6] *Core Protocols in the Internet Protocol Suite* .
http://tools.ietf.org/id/draft-baker-ietf-core-04.html .

[OMG:DDS-SECURITY] [7] Object Management Group (OMG). *DDS Security.* 2014 .
http://www.omg.org/spec/DDS-SECURITY/ .

Requests for correction

[rfc-1] obrazek

[rfc-1] obrazek