DDS通信原理分析

作 者：周 友 军

《版权申明，本文仅作为个人学习使用，不承担任何商用和法律责任》

**关 键 字**：DDS、ROS、RTPS、eProsima

**摘 要**：

**缩 略 语**：

|  |  |  |
| --- | --- | --- |
| 缩略语 | 英文全拼 | 中文释义 |
| **OSRF** | **O**pen **S**ource **R**obotics **F**oundation. | 开源机器技术基金，一个非营利性的组织进行维护，号召有能力的人对该基金进行捐款。 |
| **ROS** | **R**obot **O**perating **S**ystem. | 机器人操作系统。 |
| **DCPS** | **D**ata-**C**entric **P**ublish-**S**ubscribe | 一种基于数据为中心的发布/订阅模式。 |
| **OMG** | **O**bject **M**anagement **G**roup | 一个国际化的，允许成员自由加入的非营利性技术组织。 |
| **IDL** | **I**nterface **D**escription **L**anguage | OMG定义的一种接口描述语言。 |
| **RTI** | **R**eal-**T**ime **I**nnovations | OMG主要成员，DDS重要供应商。 |
| **DDSI-RTPS** | **DDS**-**I**nteroperability **R**eal **T**ime **P**ublish **S**ubscribe | 协同DDS实时发布-订阅。 |
|  |  |  |

# 前言

不知道写点什么好……

# DDS通信原理介绍

## DDS介绍[[1]](#footnote-1)

DDS最初被设计出来的目的是为了改进ROS1.x的传输系统（transport），以及支持新的组件库（比如ZeroMQ、Protocol Buffers、zeroconf）。

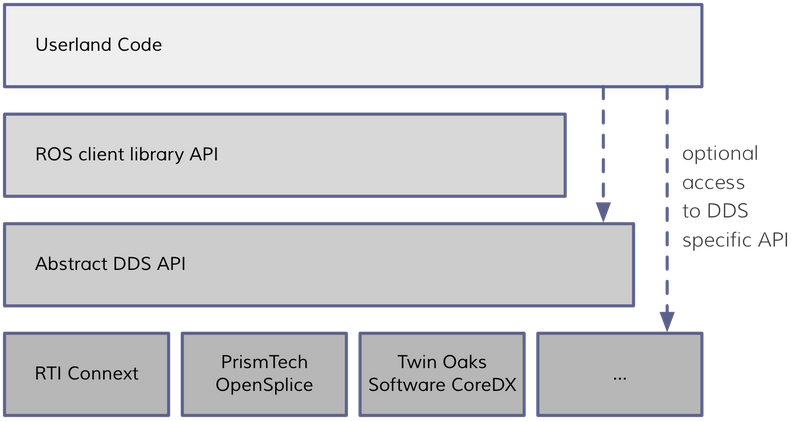
DDS提供了一种类似于ROS1.x的“发布-订阅”传输系统，它使用IDL接口描述语言来定义、序列化[[2]](#footnote-2)消息，同时也支持”request-response”风格的传输系统（在beta2版本中被称作DDS-RPC）。DDS提供的默认发现（Discovery）机制使用了它自己的“发布-订阅”传输系统，它是一个分布式的发现机制，允许任意两个DDS程序之间通信，而不像ROS1.x那样需要额外的Master节点。动态发现机制并不做强制要求，许多的DDS供应商提供静态发现机制[[3]](#footnote-3)。

刚开始的时候，一系列公司各自开发了类似的中间件，他们的公共客户期望供应商之间能够能加友好的沟通和交流，于是DDS就逐渐的成为了一个标准。该标准由OMG组织提出和创建，DDS技术是可以被完全信任的，目前在下面的几个领域都得到了广泛的使用，包括：军舰、大型基建（比如水坝）、金融系统、航天系统、太空领域、火车配电系统等。

DDS主要的供应商有：RTI、PrismTech、Twin Oaks Software[[4]](#footnote-4)。

## DDS在ROS2.x中的应用简述

DDS提供发现、消息定义、消息序列化、发布-订阅传输。其中发现、消息定义、消息序列化（至少是基础实现），ROS2.x会为基于DDS提供一组类似于ROS1.x的接口给ROS1.x主要客户，也允许用户直接访问DDS实现层功能接口。



1. DDS and ROS API Layout, http://design.ros2.org/articles/ros\_on\_dds.html

如果用户需要直接访问DDS实现层的功能接口，用户最好通过依赖一个额外的软件包，来告诉自己和某个特定的DDS供应商有依赖（有利于软件包的管理）。一旦用户字节调用了DDS实现层，将会失去支持其他DDS供应商的便利性，一般情况下不建议应用程序直接调用DDS实现层。

##### 发现 Discovery

DDS能够完全支持基于Master的ROS1.x发现系统，ROS1.x需要利用DDS API来获取节点列表信息、Topic列表信息、以及节点间的连接方式等，从而避免ROS1.x直接访问DDS实现层接口。

DDS发现系统的优势在于，它是完全的分布模式，不存在错误处理中心；同时允许用户在发现系统中自定义关键数据，在发布-订阅模式上封装更高层级的概念。

##### 发布Publish-订阅 Subscribe

DDSI-RTPS协议能够完全替代ROS1.x基于TCPROS/UDPROS协议的发布-订阅系统。相对于ROS1.x来说，DDS API提供了更多的发布-订阅模式。ROS1.x中节点等同于DDS中的graph participant，一个participant可以拥有多个Topic[[5]](#footnote-5)。

DDS中的publisher、subscriber并不会直接向topic中写入/读取数据，而是通过新增加的DataWriters、DataReaders和消息类型绑定，从而向topic中写入/读取数据。其中DataWriters由publisher创建，DataReaders由Subscriber创建。这些附加的层级设置能够更加友好的支持QoS设定[[6]](#footnote-6)。

##### 更加高效的传输选择 Efficient Transport Alternatives

在ROS1.x的实现中，并没有真正标准的基于共享内存的传输方案，因为相比较于使用localhost的TCP loop-back连接，基于共享内存的传输方法传输速度提升极其有限，如果需要有特定的应用场景特别关心共享内存带来的零拷贝收益，ROS1.x提供了Nodelets方案。

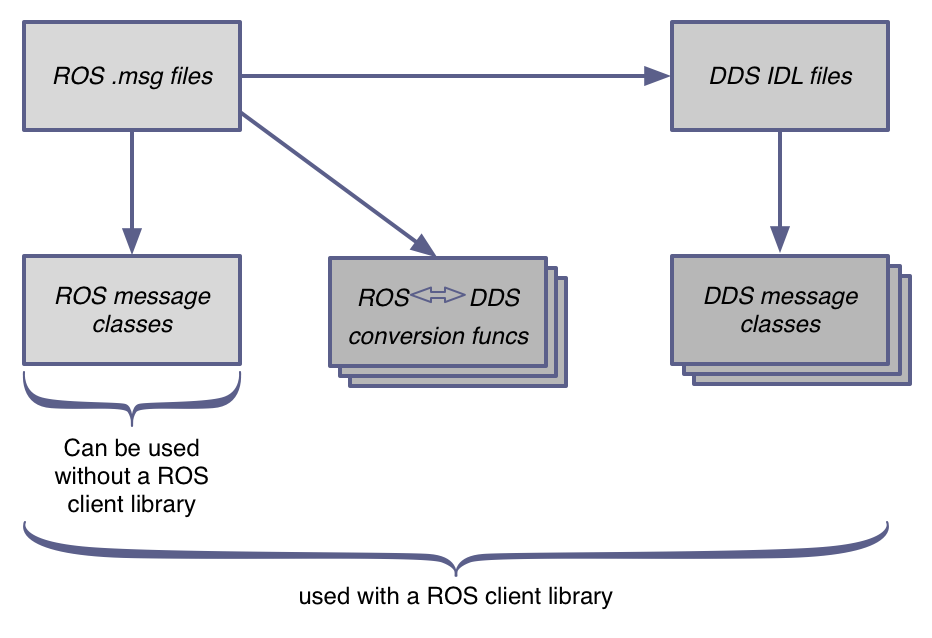
在DDS的使用场景中，大多数的供应商都会使用共享内存来优化本地消息传输效率（另外他们只使用点对点数据传输协议和UDP Sockets来进行远程消息通信），通过共享内存为DDS消息传输性能带来一个极大的提升，然而同样的实现方式在ROS1.x中并不能带来很好的效果[[7]](#footnote-7)。

然而并不是所有的DDS供应商都在这方面做了流程优化，他们可能通过topic直接将共享内存的地址进行分享，比如基于ZeroMQ的客户中间件就可能使用这种方式[[8]](#footnote-8)。

##### 消息Message

ROS定义了大量的消息值，消息格式比较简单，而且在机器人通信领域经历了多年的演进和使用。目前大量的ROS代码都是基于这些消息的结构和定义来编写的，因此为了保留消息格式和消息陈述的内容，ROS2.x保留了ROS1.x的消息定义和内容陈述方式。

ROS1.x使用了”.msg”文件来定义消息，在ROS2.x中”.msg”文件将继续被使用，同时它将会被转化为”.idl”文件供DDS使用。ROS2.x API在消息publish之前，可以在内存中专门唯一地使用”.msg”对象，但是在publish的时候需要转化为”.idl”对象。



1. Message Structure, http://design.ros2.org/articles/ros\_on\_dds.html

在”.msg”对象转化为”.idl”对象的时候，进行field-to-field拷贝看上去有很大的性能消耗，但是和消息序列化（serialization）时进行转化的代价相比，性能的消耗就显得相对的微小了。

##### 服务和请求Service and Actions

DDS目前并不为request-response style RPC提供一个受批准的、可执行的标准，来实现类似于ROS1.x中的服务[[9]](#footnote-9)功能。目前有一个RPC规格正在被OMG DDS工作组评估，还未被正式接纳和发布，部分DDS供应商已经基于此规格的草稿实现了RPC API接口。目前还不清楚该规格是否能够工作的很好，可以确认的是它能够支持non-preemptable非抢占式ROS版本。

ROS2.x目前已经弱化了服务和请求模式（成了非主流），或许在以后的进化道路上慢慢的消亡。

##### 语言支持Language Support

典型的DDS供应商会提供至少C、C++和Java的编程接口。目前还没有任何已确认的DDS版本来使用Python。

ROS2.x第一优先提供C语言版本的接口，提供所有的功能特性，因为C是最容易被Python、Ruby、Lisp等语言封装。实际操作中ROS2.x提供C/C++ DDS API，其中C++可以被封装为C，然后C被封装为其它语言格式。

## DDS规格1.4版本解读

## RTPS规格2.2版本解读

# DDS通信应用样例

# eProsima实现分析

# eProsima性能测试

本章节主要基于ROS2.x release-latest分支，而不是master分支，对eProsima Fast-RTPS进行性能测试和评估。

用户可以在网站”http://[www.eprosima.com/index.php/resources-all/performance/40-eprosima-fast-rtps-performance](http://www.eprosima.com/index.php/resources-all/performance/40-eprosima-fast-rtps-performance)”上查看eProsima官方基于”Intel Core [i3@3.4GHz](mailto:i3@3.4GHz), 4GB RAM, Intel Gigabit Network adapter at 1Gbps”公布的性能数据，提供了时延、吞吐量基本性能数据。

我们用于性能测试评估的服务器为”RH2288H V3”，基本硬件信息为：

* **处理器**(x2)
  + Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.2GHz
  + 12 cores/24 threads
  + L1/L2/L3 Cache, 21/256/30720KB
* **内存**(x8)：
  + 16GB/DIMM @2.4GHz
* **网口[[10]](#footnote-10)**：
  + Intel Corporation 82599ES 10-Gigabit SFI/SFP+

## 供应商测试用例分析

供应商eProsima本身也提供和发布了一系列的测试用例，所有的用例可以在”https：//github.com/eProsima/Fast-RTPS/tree/master/test”地址获取。

## 通信性能测试

面向Distributed Participant 的传输性能测试，Publisher和Subscriber可以部署在本地Localhost上，也可以分别部署在不同的网络设备上。

X86 Server

X86 Server

GE Wired Connection

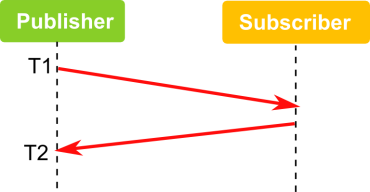
1. GE Wired Network连接示意图

由于测试环境原因，当前阶段尽在GE Wired Connection方式下进行测试，后续根据机房的实际情况单独部署LAN SWITCH交换机的场景进行测试[[11]](#footnote-11)。

提前将所有的原始测试数据附加在此处。

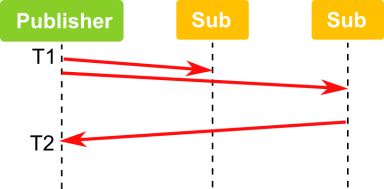
##### 时延Latency

时延测试中一般进行单程时延测试、全程时延测试。在RTPS通信中，单程时延定义为Publisher发送消息🡪 Subscriber接收到消息的时间；全程时延定义为Publisher发送消息🡪 Subscriber接收消息，消息回发同一个Publisher 🡪 Publisher接收到回发的消息，如下图所示。



1. 单Subscriber场景下RTPS全程时延示意图

在多个Subscriber的场景，测试方法类似。在这种场景下，Publisher发送数据到多个Subscriber，但是只有一个Subscriber回发消息给Publisher。



1. 多Subscriber场景下的RTPS全程时延示意图

时延计算公式：

***Latency\_round-trip = (T2 – T1)/2***

1. Localhost本地通信时延数据(1 Publisher vs. 1 Subscriber)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Topic Latency Performance Raw Data in us** (Localhost Transport, 1p1s) | | | | | |
| Message size (bytes) | Samples | variance | Avg. | min | max |
| 16 | 100,000 | 17.00 | 40.75 | 32.26 | 2119.46 |
| 32 | 100,000 | 21.00 | 41.70 | 31.00 | 2112.49 |
| 64 | 100,000 | 11.00 | 35.60 | 32.83 | 1628.55 |
| 128 | 100,000 | 19.00 | 40.45 | 34.02 | 2039.65 |
| 256 | 100,000 | 14.00 | 37.48 | 31.94 | 2608.43 |
| 512 | 100,000 | 10.00 | 35.17 | 32.34 | 1633.61 |
| 1,024 | 100,000 | 11.00 | 35.97 | 33.07 | 1550.98 |
| 2,048 | 100,000 | 28.00 | 50.12 | 35.20 | 4159.05 |
| 4,096 | 100,000 | 14.00 | 43.33 | 38.17 | 1550.94 |
| 8,192 | 100,000 | 15.00 | 52.79 | 46.00 | 1593.51 |
| 16,384 | 100,000 | 30.00 | 73.72 | 59.38 | 2351.46 |

1. Localhost本地通信时延数据(1 Publisher vs. 20 Subscriber)

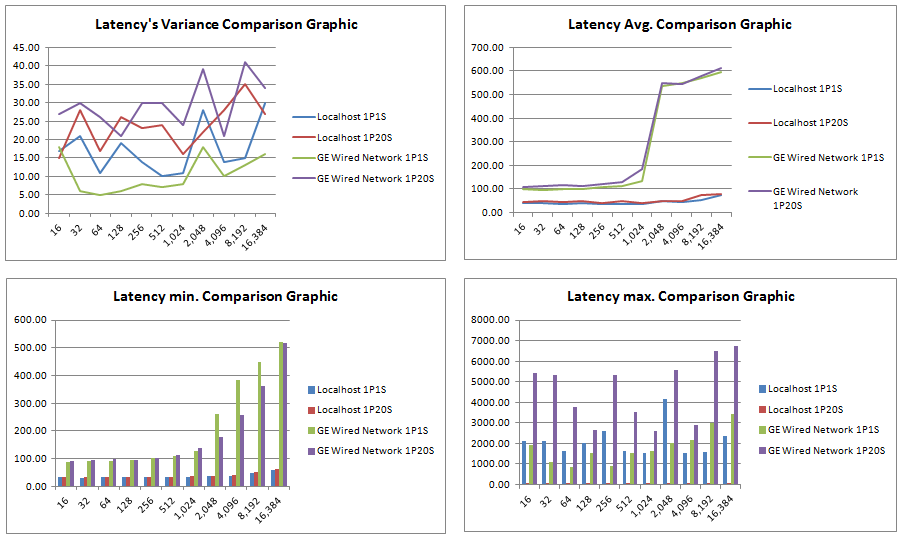
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Topic Latency Performance Raw Data in us (Localhost Transport, 1p20s)** | | | | | |
| Message size (bytes) | Samples | variance | Avg. | min | max |
| 16 | 100,000 | 15.00 | 43.21 | 34.27 | 2049.50 |
| 32 | 100,000 | 28.00 | 47.80 | 34.04 | 3541.35 |
| 64 | 100,000 | 17.00 | 44.83 | 33.27 | 2040.70 |
| 128 | 100,000 | 26.00 | 47.39 | 33.60 | 2630.85 |
| 256 | 100,000 | 23.00 | 42.22 | 34.32 | 2310.02 |
| 512 | 100,000 | 24.00 | 48.53 | 34.98 | 2236.66 |
| 1,024 | 100,000 | 16.00 | 40.22 | 35.80 | 1997.29 |
| 2,048 | 100,000 | 22.00 | 49.43 | 37.35 | 2291.25 |
| 4,096 | 100,000 | 28.00 | 47.25 | 41.56 | 3922.12 |
| 8,192 | 100,000 | 35.00 | 73.52 | 50.20 | 4289.99 |
| 16,384 | 100,000 | 27.00 | 78.74 | 62.89 | 3101.13 |

1. GE Wired Network网线直连时延数据(1 Publisher vs. 1 Subscriber)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Topic Latency Performance Raw Data in us (Network Transport, 1p1s)** | | | | | |
| Message size (bytes) | Samples | variance | Avg. | min | max |
| 16 | 100,000 | 18.00 | 100.58 | 87.63 | 1919.14 |
| 32 | 100,000 | 6.00 | 96.37 | 91.70 | 1081.02 |
| 64 | 100,000 | 5.00 | 98.70 | 92.67 | 850.77 |
| 128 | 100,000 | 6.00 | 98.96 | 95.93 | 1512.54 |
| 256 | 100,000 | 8.00 | 106.69 | 100.28 | 903.23 |
| 512 | 100,000 | 7.00 | 113.89 | 108.85 | 1543.71 |
| 1,024 | 100,000 | 8.00 | 133.23 | 127.56 | 1630.57 |
| 2,048 | 100,000 | 18.00 | 535.61 | 260.43 | 2032.27 |
| 4,096 | 100,000 | 10.00 | 546.71 | 384.00 | 2166.00 |
| 8,192 | 100,000 | 13.00 | 569.89 | 447.34 | 3004.55 |
| 16,384 | 100,000 | 16.00 | 595.51 | 520.13 | 3417.32 |

1. GE Wired Network网线直连时延数据(1 Publisher vs. 20 Subscriber)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Topic Latency Performance Raw Data in us (Network Transport, 1p1s)** | | | | | |
| Message size (bytes) | Samples | variance | Avg. | min | max |
| 16 | 100,000 | 27.00 | 107.93 | 89.31 | 5440.68 |
| 32 | 100,000 | 30.00 | 114.18 | 95.89 | 5308.23 |
| 64 | 100,000 | 26.00 | 115.66 | 97.54 | 3765.97 |
| 128 | 100,000 | 21.00 | 112.45 | 95.61 | 2669.69 |
| 256 | 100,000 | 30.00 | 119.88 | 102.41 | 5309.50 |
| 512 | 100,000 | 30.00 | 129.44 | 111.28 | 3523.78 |
| 1,024 | 100,000 | 24.00 | 185.52 | 136.34 | 2590.18 |
| 2,048 | 100,000 | 39.00 | 547.87 | 178.21 | 5550.87 |
| 4,096 | 100,000 | 21.00 | 544.15 | 256.12 | 2884.97 |
| 8,192 | 100,000 | 41.00 | 577.37 | 361.58 | 6496.34 |
| 16,384 | 100,000 | 34.00 | 612.58 | 518.02 | 6736.16 |



1. 时延对比图(X-Message Size in Bytes, Y-Latency in us)

根据均值、方差进行正态分布图标绘制的结果如下：

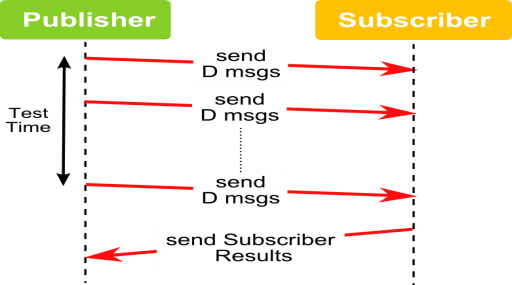
1. 时延正态分布图标

（备注：时延测试采用的是eProsima提供的LatencyTest工具，样本个数为100,000，此工具在连续测试不同message size的时候存在bug，因此在测试的时候做了适量的代码适配，每个message size测试单独进行。）

##### 吞出量Throughput

在通信领域有很多测试吞吐量的方法，其中最常用的方法是发送一个很大的文件（或者很多的小文件），测量数据到达网络另外一段的时间。

在RTPS通信场景，一次性发送一组消息，计算出发出的数据大小和到达的时间。为了获取自大的吞吐量，需要尝试发送不同数量的消息（demands）来找到最佳的结果（无丢包）。



1. RTPS吞吐量测试示意图

在测试过程中，使用不同大小的Message进行测试（16B/32B/64B/…），每一个大小的Message使用不同的demands进行尝试（100/200…），找到不丢包的最大传输速率，作为其吞吐量指标。

下面是在机房测试环境（Publisher + Subscriber + 10GSwither + Other Devices），使用随ros2发布的eProsima版本进行测试的结果数据，Publisher和Subscriber分别部署在独立的服务器上面。

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **10G Network TEST** | Publisher-Subscriber Throughput in MBits/sec | | | | | | | | | |
| 16B | 32B | 64B | 128B | 256B | 512B | 1024B | 2048B | 4096B | 8192B |
| Publisher | 3.911 | 4.107 | 78.296 | 18.823 | 27.999 | 48.727 | 65.111 | 178.242 | 139.067 | 119.071 |
| Subscriber | 3.830 | 4.023 | 76.688 | 18.434 | 27.422 | 47.721 | 63.768 | 174.570 | 136.199 | 116.617 |
| **Localhost TEST** | Publisher-Subscriber Throughput in MBits/sec | | | | | | | | | |
| 16B | 32B | 64B | 128B | 256B | 512B | 1024B | 2048B | 4096B | 8192B |
| Publisher | 6.342 | 12.994 | 16.777 | 31.974 | 64.403 | 106.490 | 180.374 | 224.119 | 301.653 | 127.272 |
| Subscriber | 6.214 | 12.727 | 16.431 | 31.316 | 63.077 | 104.297 | 176.659 | 219.503 | 295.434 | 124.653 |

1. 吞吐量Throughput测试数据曲线图

参考eProsima官方发布的性能数据，在1024B大小的时候能能达到600MBits/sec的吞出量，实际测试仅有64MBits/sec左右，实测数据差异较大（正在联系机房管理员对测试设备通过网络直连后再做验证测试）。

## 本地通信性能测试Shared-memory vs. Localhost UDP Loop-back

## QoS Influence

# 附录：代码样本

# 参考文献

[1] <http://design.ros2.org/articles/ros_on_dds.html>

[2] <http://ieeexplore.ieee.org/document/7743223/?reload=true>

[3] Data Distribution Service Specification Version 1.4, <http://www.omg.org/spec/DDS>

[4] DDS Interoperability Wire Protocol Specification Version 2.2， <http://www.omg.org/spec/DDSI-RTPS>

[5] ROS DDS原型和一些实际尝试, <https://github.com/osrf/ros_dds>

[6] eProsima参考网址<http://eprosima-fast-rtps.readthedocs.io，http://eprosima-fast-rtps.readthedocs.io/en/latest/introduction.html>, <http://eprosima-fast-rtps.readthedocs.io/en/latest/requirements.html>

[7] ROS2.x Wiki，<https://github.com/ros2/ros2/wiki/Tutorials>

[8] Install ROS2.x under linux from source, <https://github.com/ros2/ros2/wiki/Linux-Development-Setup>

[9] eProsima官方性能数据，[www.eprosima.com/index.php/resources-all/performance/40-eprosima-fast-rtps-performance](http://www.eprosima.com/index.php/resources-all/performance/40-eprosima-fast-rtps-performance)

[10] eProsima Fast-RTPS github, <https://github.com/eProsima/Fast-RTPS>

1. 本章内容部分摘录并翻译自http://design.ros2.org/articles/ros\_on\_dds.html [↑](#footnote-ref-1)
2. Message Serialization目前没有很准确的翻译表达，暂时翻译为序列化。 [↑](#footnote-ref-2)
3. 发现机制（Discovery）在后续章节中单独描述和解释。 [↑](#footnote-ref-3)
4. OMG Members：OCI, RTI, Twin Oaks Computing, Inc. , PrismTech, Sparx Systems, eProsima, <http://dds-directory.omg.org/vendor/list.htm>. [↑](#footnote-ref-4)
5. Participant也可以没有topic，这里topic和ROS1.x中topic类似。 [↑](#footnote-ref-5)
6. QoS在后续的章节单独介绍。 [↑](#footnote-ref-6)
7. ROS1.x处理流程为a.消息存放到一个大buffer，b.调用tcp send一次性发送；DDS处理流程为a.消息拆分为许多udp报文 b.多次调用udp send发送；多次的发送udp报文并不能够像一次性发送一个大的tcp报文那样享受同样的速度收益，因此许多DDS供应商简化了localhost的消息处理流程，使用一个基于共享内存的机制来提升消息传输效率。 [↑](#footnote-ref-7)
8. ZeroMQ不在本文档中分析，有时间单独分析看看…… [↑](#footnote-ref-8)
9. ROS2.x更多强调的是distribution，分布式对等观念，而弱化了服务特性。 [↑](#footnote-ref-9)
10. 由于松山湖的实验室服务器设备当前仅连接了10G网口，同时也方面性能测试向上探，暂时使用10G网口来进行性能测试。 [↑](#footnote-ref-10)
11. 第一次测试使用机房公共交换机，由于复杂的应用环境导致测试数据失真。 [↑](#footnote-ref-11)