臻融数据分发服务 DDS 系统软件[1]

故障排查指南[2]

[3]

单位名称：南京臻融科技有限公司[4]
公司地址：南京市江宁区将军大道南佑路 7 号[5]
公司电话：025-52106986[6]
网 址：www.zrtechnology.com[7]
邮 编：211106[8]

目录 1

1. DDS 通信过程.....1 2
1.1 发现过程.....1
1.2 匹配过程.....3
1.3 通信过程.....3
1.3.1 尽力而为模式.....4
1.3.2 可靠模式.....4
2 排故手段.....7
2.1 网络工具.....7
2.1.1 ping.....7
2.1.2 iperf.....8
2.1.3 sockperf.....13
2.2 交互式日志.....18
2.2.1 ZRDDS 日志系统.....18
2.2.2 ZRDDS 日志控制.....20
2.2.3 ZRDDS 常用调试日志号.....23
2.3 抓包工具.....27
2.3.1 DDS 的数据包封装.....27
2.3.2 tcpdump.....28
2.3.3 WireShark.....29
2.4 ZRDDS 管理监控器.....34
3 典型故障.....38
3.1 编译失败.....38
3.1.1 环境变量未生效.....38
3.1.2 预编译符缺失.....38
3.1.3 头文件与库不匹配.....38
3.1.4 库与编译方式不匹配.....39
3.1.5 动态库缺失.....39
3.1.6 VisualGDB 库缺失.....39
3.1.7 丢失 Psapi.lib.....40
3.1.8 VS2015 版本（update 3）和库不匹配.....40
3.2 初始化失败.....41
3.2.1 报错.....41
3.2.2 崩溃.....45
3.2.3 其他.....45
3.3 收不到数据.....46
3.3.1 网络环境检测.....46
3.3.2 ZRDDS 配置检测.....47
3.3.3 高级诊断.....49
3.4 通信异常.....51
3.4.1 序列化/反序列化失败.....51
3.4.2 丢包严重.....52

3.4.3 数据中断.....52 1
3.4.4 发送失败.....52
3.4.5 网络中断.....53
3.5 异常崩溃.....54
3.6 错误日志.....57
3.6.1 日志索引.....57
3.6.2 无法找到对端可用地址.....58
3.6.3 序列化失败.....59
3.6.4 发现不一致版本.....60
3.6.5 设置优先级失败.....60
3.6.6 XML 文件解析失败.....61
3.6.7 零拷贝禁用可靠性策略.....61
3.6.8 没有可用端口.....62
4 排故流程.....62
附录 1 socket 编程常见错误码及含义.....1

表格目录 1

表格 1 通信模式匹配.....4 2
表格 2 ping 工具的常见用法.....7
表格 3 iperf 的常见用法.....8
表格 4 Iperf 的使用示例.....9
表格 5 sockperf 的常见用法.....13
表格 6 sockperf 的使用示例.....14
表格 7 调试命令编号与参数对照表.....21
表格 8 特殊主题与含义对照表.....21
表格 9 使能级别与参数值对照表.....22
表格 10 ZRDDS 常用调试日志号.....23
表格 11 tcpdump 常用命令.....29
表格 12 子消息分析常用字段.....33

图目录 1

图 1 发现信息丢失.....2 2
图 2 单播优化.....3
图 3 可靠模式通信.....5
图 4 可靠模式下的数据覆盖.....6
图 5 可靠模式下的数据发送失败.....7
图 6 ping 工具的使用结果示例.....8
图 7 UDP 测试 Server 端.....10
图 8 UDP 测试 Client 端.....11
图 9 TCP 测试 Server 端.....11
图 10 TCP 测试 Client 端.....12
图 11 Multicast 测试 Server 端.....12
图 12 Multicast 测试 Client 端.....13
图 13 UDP 测试 Client 端.....15
图 14 UDP 测试 Server 端.....16
图 15 TCP 测试 Client 端.....16
图 16 TCP 测试 Server 端.....17
图 17 Multicast 测试 Client 端.....17
图 18 Multicast 测试 Server 端.....18
图 19 RTPS 消息结构.....27
图 20 子消息结构.....28
图 21 选择抓取网卡.....30
图 22 开始捕获.....30
图 23 停止捕获.....30
图 24 数据分析图.....31
图 25 过滤后数据.....31
图 26 Wireshark 解析 DDS 数据包.....32
图 27 RTPS 消息详情.....32
图 28 系统物理视图.....35
图 29 进程详细信息.....35
图 30 系统逻辑视图.....36
图 31 主题详细信息视图.....37
图 32 匹配分析视图.....37
图 33 预编译符缺失导致的编译失败报错图.....38
图 34 缺失 pthread 报错图.....39
图 35 缺失 dl 库报错图.....40
图 36 licence 文件未找到.....41
图 37 licence 文件被修改.....41
图 38 licence 文件过期.....41
图 39 没有可用网卡.....43
图 40 没有网卡.....44
图 41 程序中的 DDS 版本信息打印.....45
图 42 日志文件中的 DDS 版本信息打印.....45

图 43 销毁信号量.....46 1
图 44 创建域参与者.....47
图 45 监控器域视图.....47
图 46 创建主题.....48
图 47 主题信息.....48
图 48 创建数据写者.....48
图 49 Qos 不匹配打印.....48
图 50 实体 Qos 信息视图.....49
图 51 配置域参与者使用的地址.....49
图 52 DATA 子消息.....50
图 53 HEARBEAT 子消息.....50
图 54 ACKNACK 子消息“ACK”.....50
图 55ACKNACK 子消息“NACK”.....51
图 56 序列化失败.....51
图 57 反序列化失败.....52
图 58 发送队列已满.....53
图 59 发送失败（超时）.....53
图 60 尝试重连.....53
图 61 重连失败.....53
图 62 重连成功.....53
图 63 组播发送失败.....54
图 64 程序崩溃图.....55
图 65 GDB 加载 core 文件.....56
图 66 序列化 sequence 数据失败.....59

1. DDS 通信过程 1

在默认的典型配置之下，DDS 是基于 UDP 实现的应用层通信协议。DDS 在 UDP 的简单协议之上，对用户数据进行了封封装，实现了多种功能的支持。以下分析均基于使用默认配置的 DDS 应用进行。 2

DDS 从启动到通信通常需要经历三个过程，分别是发现、匹配和通信。这三个过程必须顺序进行，后一个过程必须依赖前一个过程的信息。 3

1.1 发现过程 4

DDS 的发现是通过 DomainParticipant 向特定的组播地址发送数据并接收来自其他 DomainParticipant 的组播数据实现的。在 DDS 发送的发现数据中，包含了 DomainParticipant 的基本信息，包括其 GUID、数据接收地址和端口、超时时间等信息。通过这些信息，可以对 DomainParticipant 建立起单播通信的通道，进行后续的过程。 5

DomainParticipant 会按照一定的频率发送发现数据，通常发送的间隔小于其声明的超时时间。如果超过了超时时间未收到某个 DomainParticipant 发送的发现数据，该 DomainParticipant 将被判定为下线，与之相关的所有信息都将被清除。DomainParticipant 也可以主动发送下线的数据包，表明自己将要离开。但是由于该数据并没有可靠传输的协议保证其能够被收到，因此不应当依赖这一过程。对于其他 DomainParticipant 而言，收到主动发送的数据包而下线与因超过设定时间未收到发现数据包导致下线的过程是一样的，仅在时间长短上有区别。 6

在 DDS 协议中，仅要求组播发送发现数据。在实现中，为了加快发现速度，在收到新的 DomainParticipant 发送的发现数据后，会立即通过单播向其发送自身的发现数据，以减少发现所需时间。 7

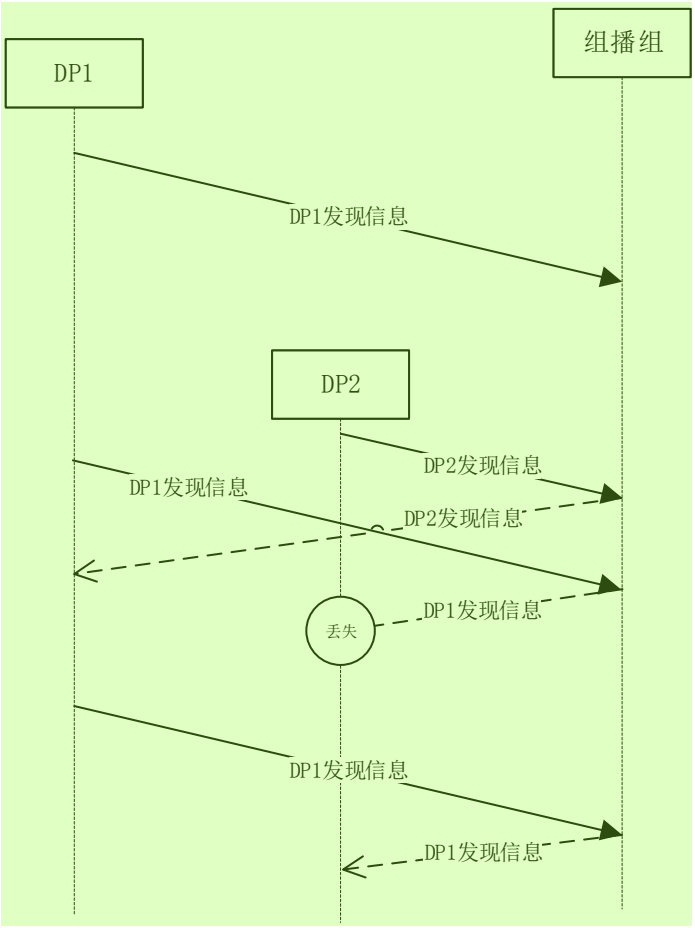


图 1 发现信息丢失

如图 1 所示，当 DP1 发出的发现信息丢失后，DP2 无法在其创建后立即发现 DP1，而需要等待 DP1 下一个发现信息的发送周期时才能发现。若发现信息发送频率较低，会明显延长发现的时间消耗。

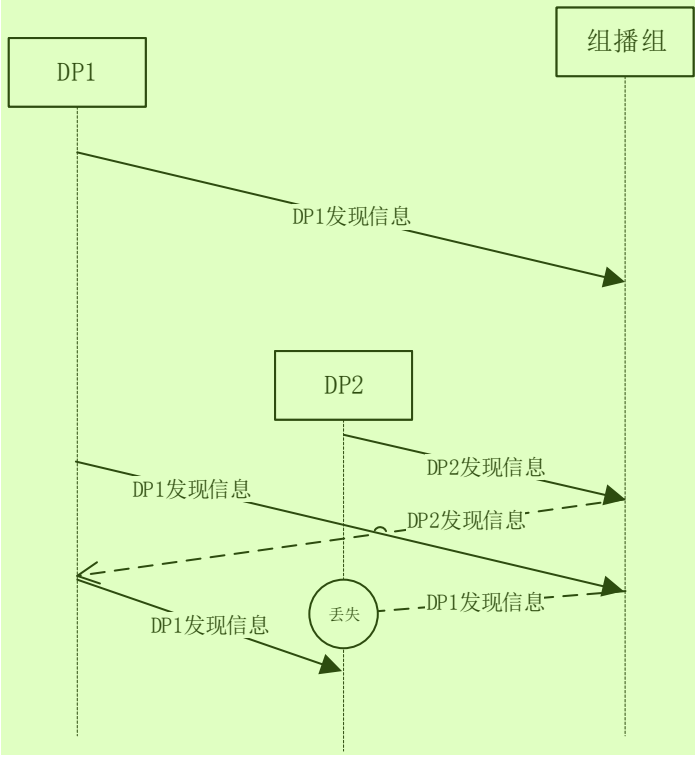


图 2 单播优化

当加入单播优化之后，如图 2 所示，当 DP1 收到 DP2 发出的发现信息后，立即通过单播向 DP2 发送自身的发现信息，使 DP2 也能及时发现 DP1，可以明显缩短因为发现信息的偶然丢失产生的发现延迟。

1.2 匹配过程

在 DomainParticipant 互相发现后，就可以进行匹配过程。

DDS 的匹配过程是指在两个 DomainParticipant 之间交互用户创建的主题、数据写者和数据读者的信息，再根据对方的信息与自身的信息建立同一主题下数据写者和数据读者的关联关系。

匹配的数据交互过程都是通过单播进行。匹配双方会通过内置主题发送自身的数据读者和数据写者的信息，包括 GUID、所属主题、所属类型、需要一致性判断的 QoS 等。收发这些数据的数据读者和数据写者都被设置为可靠通信模式，因此能够保证这些信息不会因网络丢包等原因丢失。当收到对方的数据读者和数据写者信息后，首先检查自身是否包含对应的主题，然后在相应主题下将对方的数据写者与自身的数据读者、对方的数据读者与自身的数据写者进行一一配对，配对过程需要检查 QoS 是否能够兼容，最后就可以在能够匹配的数据写者和数据读者之间建立数据通路，开始通信过程。

1.3 通信过程

DDS 通信过程可以分成两个模式，分别为尽力而为（Best-effort）和可靠（Reliable）。使

用哪一种模式取决于数据写者和数据读者的 QoS 配置，具体情况如表格 1 所示。1

表格 1 通信模式匹配 2

数据写者配置	数据读者配置	实际使用
尽力而为	尽力而为	尽力而为
尽力而为	可靠	不匹配，无法通信
可靠	尽力而为	尽力而为
可靠	可靠	可靠

3

后面说的通信模式都是指实际使用的模式。4

1.3.1 尽力而为模式 5

在尽力而为模式下，数据能否送达取决于网络层的可靠性。DDS 使用 UDP 进行数据收发，而 UDP 本身并不提供可靠机制，有可能产生数据丢失、乱序的问题。DDS 会对发出的每一个数据包编号，并在接收端（数据读者）按编号顺序提交数据。因此，DDS 可以保证数据读者接收到的数据不会产生乱序。但是，当数据包丢失时，DDS 会直接将其跳过，不会采取重传等措施，因此当网络层发生数据丢失或因为缓存填满导致数据被丢弃时，对用户而言就发生了丢包。 6

对于发送端（数据写者）而言，数据读者不会反馈任何信息，因此发出的数据是否被数据读者收到是未知的。 7

由于尽力而为模式不会因为数据丢失、乱序导致额外的开销，因此其收发速度和资源占用优于可靠模式。对于没有可靠性要求的数据——例如传感器周期性上报的信息——可以采用尽力而为模式发送。 8

1.3.2 可靠模式 9

DDS 的可靠模式通过心跳-反馈的方式实现。当用户提交数据到数据写者之后，数据写者会对数据进行编号，在编号数值溢出前，是严格逐个递增的。数据读者在收到数据之后，就能够根据其编号缺失的情况确定是否发生了丢包，并根据数据写者发送的心跳数据确认头尾缺失的数据。数据读者确定数据丢失的情况后，会通过反馈的方式通知数据写者，令其重新发送丢失的数据。 10

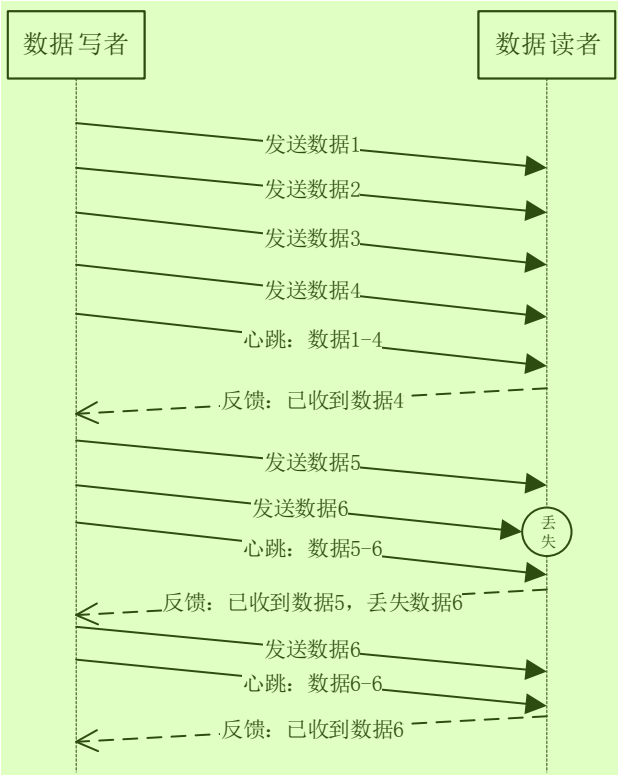
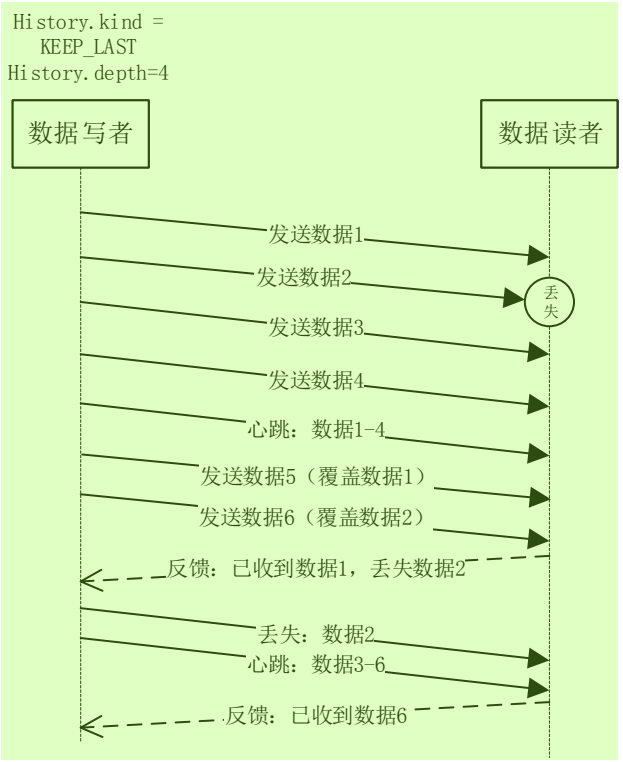


图 3 可靠模式通信

如图 3 所示，数据写者将数据编号后发送到数据读者，并定时发送心跳数据，告知数据读者当前的数据范围。数据读者收到数据和心跳后，会向数据写者反馈自己接收的情况。若数据读者发生了数据丢失，将会通过反馈告知数据写者，数据写者根据反馈重新发送丢失的数据。

为了能够支持数据重发，要求数据写者能够保留一部分用户数据。保留数据的策略通过 HistoryQos 进行配置，HistoryQos 中，包含的 kind 成员定义数据应当保留最新（KEEP_LAST）还是完整保留（KEEP_ALL）。当设置为保留最新的策略时，数据写者仅保留 HistoryQos.depth 中设置的数据数量（暂不考虑实例）。当数量超出这个限制时，数据写者会直接将最旧的数据覆盖。如果此时被覆盖的数据没有被数据读者收到，仍然会产生数据丢失。

其过程如图 4 所示。数据写者被设置为 KEEP_LAST 策略，且数据样本仅保留 4 个。当其连续发送数据时，无论数据读者是否收到（即数据写者是否收到数据读者的反馈信息），都会将旧的数据覆盖：当发送数据 5 时覆盖数据 1，当发送数据 6 时覆盖数据 2。如果数据 2 恰好丢失，当数据读者的反馈到达数据写者时，数据 2 已经被覆盖，无法重发。此时数据写者就会告知数据读者数据 2 已经丢失，并重新发送心跳信息，指明当前数据范围为 3-6。数据读者收到这一信息后，确认 2 无法重传，发生了数据丢包，然后反馈其余数据包已经收到。



1

图 4 可靠模式下的数据覆盖 2

如果用户不希望出现数据覆盖的情况，应当将数据写者历史数据的保存策略设置为 KEEP_ALL。此时 HistoryQos.depth 的值将不再起效，数据写者会按照 ResourceLimits 中的设置保留尽可能多的历史数据。当历史数据的数量超出限制且数据读者未反馈时，数据写者会阻塞调用者或者返回失败。 3

如图 5 所示，数据写者的历史数据保存策略设置为 KEEP_ALL 时，若数据读者未反馈其接收状态，则在保存的数据数量超出限制时，将返回失败，直到数据读者反馈后才能将数据覆盖。需要注意的一点是当用户通过数据写者发送数据失败时，数据的编号并不会增加，直到数据发送成功后才增加。 4

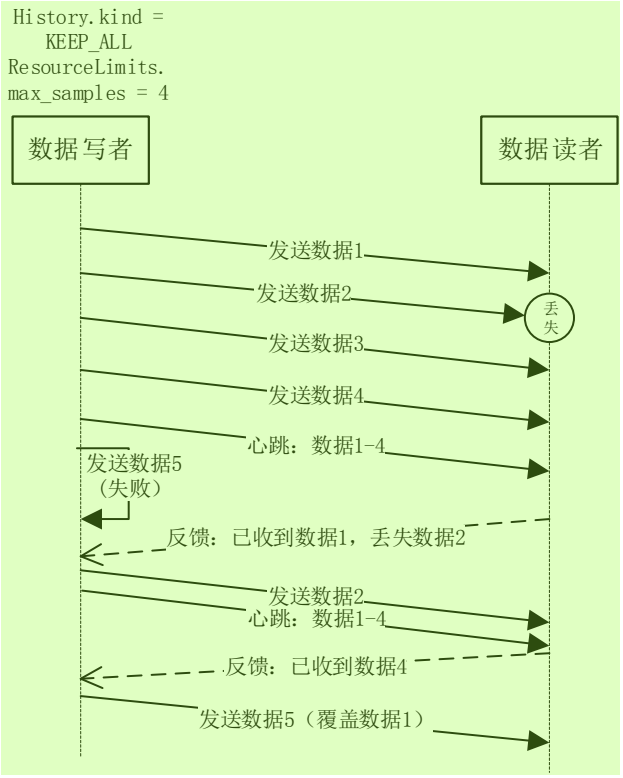


图 5 可靠模式下的数据发送失败

虽然此处描述的通信是用户主题数据的通信，但是 DDS 内置主题也是按照相同的方式进行处理，因此在分析其过程时，也可以按照同样的过程分析。

2 排故手段

2.1 网络工具

2.1.1 ping

ping 命名可以检测两台机器之间网络是否联通，详细用法可通过 ping –h 指令进行了解，此处只介绍部分常见用法。

常见用法：

表格 2 ping 工具的常见用法

ping -h	查看详细使用方式
ping <IP>	检测是否能与地址为 IP 的设备通信

示例：

检测是否能与 192.168.31.11 的设备通信

1

图 6 ping 工具的使用结果示例 2

2.1.2 iperf 3

iperf 是一个网络性能测试用具，可以测试 tcp 和 udp 带宽质量。详细用法可通过 iperf –h 4 指令进行了解，此处只介绍部分常见用法。

常见用法: 5

表格 3iperf 的常见用法 6

7

指令	含义	示例
公用选项		
-h	查看使用指南	iperf -h
-B	绑定一个主机地址或接口（用于多网卡多 IP 情况下指定所使用的 IP）	iperf -s -B 192.168.31.11
-u	指定使用 udp 协议	
-p	指定服务端使用的端口或客户端所连接的端口	iperf –c 192.168.31.11 –p 6001
-i	指定每次报告之间的间隔时间,单位为 s，默认值为 1	iperf –c 192.168.31.11 –i 2
-w	指定 TCP 窗口大小	iperf –c 192.168.31.11 -w 1M
server 专用选项		

-s	作为 server 端启用	iperf -s
client 专用选项		
-c	作为 client 端启用	iperf –c 192.168.31.11 (192.168.31.11 为 server 端地址)
-F	指定文件作为数据流进行带宽测试	iperf –c 192.168.31.11 –F test.tar.gz
-t	测试时间	iperf –c 192.168.31.11 -t 60

2

注 1：当测试时，使用 udp 协议时，收发端都要指定相同的带宽，包长；当使用 tcp 时不用指定带宽，指定 host 就可以了；3

注 2：带宽： 40g 40000M 千兆 1000M 4

注 3：当某些数据包过大需要指定 tcp 窗口大小(默认 85k)；5

注 4：udp 发不了超过 8M 的数据，会因为 sample too long 而 write failed. 6

例如： server: 7

iperf –u –s –B [本机 ip] 8

client: 1

udp: iperf –u –c 192.168.31.11(server 端的 ip) –b 1000M –l 1024

tcp: iperf –c 192.168.31.11 –l 1024

注：9

1.吞吐量参照 server 端的 BandWidth 或者是 client 端的 server Report 的 bandWidth 10

2.server 端根据情况绑定 ip 地址和带宽（udp）；11

3.tcp 测试时延加-N(no delay) 12

4. 设置包长（-l）13

示例：14

按照示例指令在两个节点间分别启用客户端和服务端，可进行相应模式下的连通性测试。15

表格 4Iperf 的使用示例 16

iperf 收发示例		
模式	Client	Server
udp	iperf -u	iperf -u

17

	-c <serverIP> -p <ServerPort>	-s -B <serverIP> -p <ServerPort>
tcp	iperf -c <serverIP> -p <ServerPort>	iperf -s -B <serverIP> -p <ServerPort>
multicast	iperf -u -c <multicastIP> -p <multicastPort>	iperf -u -s -B <multicastIP> -p <multicastPort>
参数值		
<multicastIP> <multicastPort> <serverIP> <serverPort>	组播地址 组播端口 服务端监听地址 服务端监听端口	

1

结果示例： 2

UDP 测试：3

4

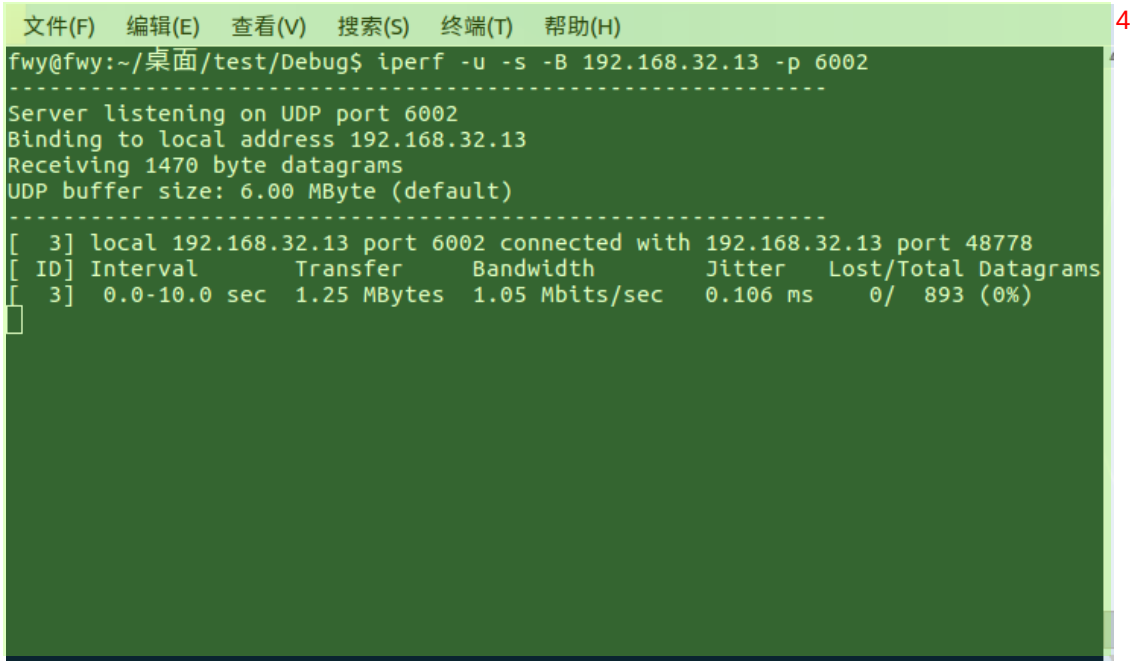


图 7 UDP 测试 Server 端 5

1

```
文件(F)  编辑(E)  查看(V)  搜索(S)  终端(T)  帮助(H)
fwy@fwy:~/桌面/test$ iperf -u -c 192.168.32.13 -p 6002
------------------------------------------------------------
Client connecting to 192.168.32.13, UDP port 6002
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 192.168.32.13 port 48778 connected with 192.168.32.13 port 6002
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[  3] Sent 893 datagrams
[  3] Server Report:
[  3]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec   0.106 ms    0/  893 (0%)
fwy@fwy:~/桌面/test$
```

图 8UDP 测试 Client 端 2

TCP 测试: 3

4

```
文件(F)  编辑(E)  查看(V)  搜索(S)  终端(T)  帮助(H)
fwy@fwy:~/桌面/test/Debug$ iperf -s -B 192.168.32.13 -p 6002
------------------------------------------------------------
Server listening on TCP port 6002
Binding to local address 192.168.32.13
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  4] local 192.168.32.13 port 6002 connected with 192.168.32.13 port 58852
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec  59.0 GBytes  50.7 Gbits/sec
```

图 9 TCP 测试 Server 端 5

1

```
文件(F)  编辑(E)  查看(V)  搜索(S)  终端(T)  帮助(H)
fwy@fwy:~/桌面/test$ iperf -c 192.168.32.13 -p 6002
------------------------------------------------------------
Client connecting to 192.168.32.13, TCP port 6002
TCP window size: 2.50 MByte (default)
------------------------------------------------------------
[  3] local 192.168.32.13 port 58852 connected with 192.168.32.13 port 6002
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  59.0 GBytes  50.7 Gbits/sec
fwy@fwy:~/桌面/test$
```

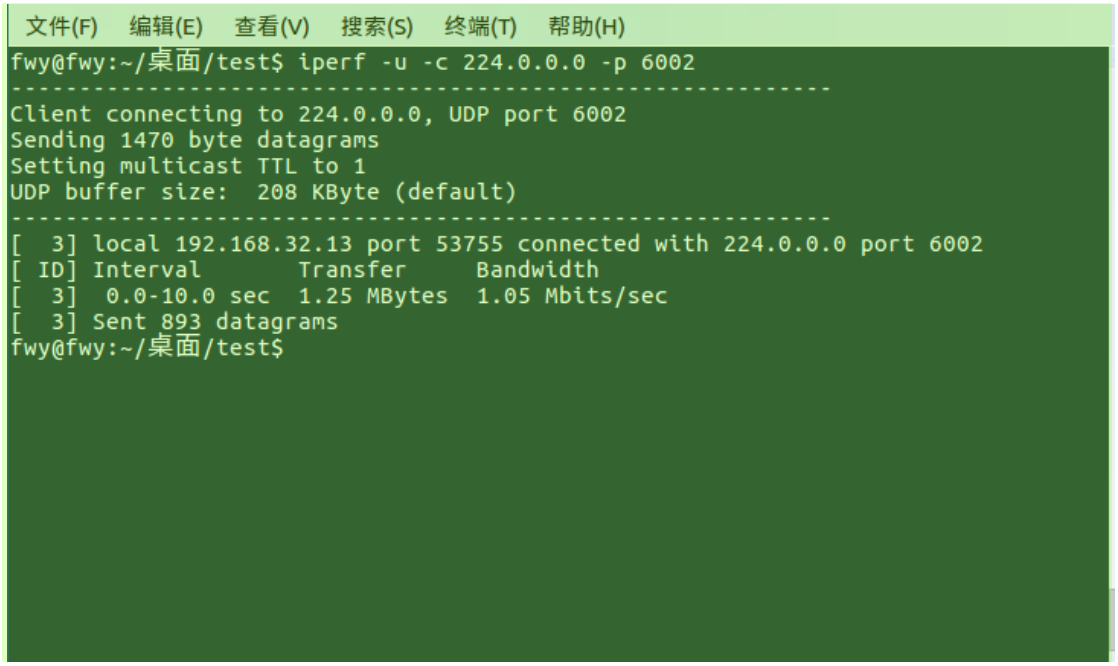
图 10TCP 测试 Client 端 2

Multicast 测试：3

4

```
文件(F)  编辑(E)  查看(V)  搜索(S)  终端(T)  帮助(H)
fwy@fwy:~/桌面/test/Debug$ iperf -u -s -B 224.0.0.0 -p 6002
------------------------------------------------------------
Server listening on UDP port 6002
Binding to local address 224.0.0.0
Joining multicast group  224.0.0.0
Receiving 1470 byte datagrams
UDP buffer size: 6.00 MByte (default)
------------------------------------------------------------
[  3] local 224.0.0.0 port 6002 connected with 192.168.32.13 port 53755
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec   0.104 ms    0/  893 (0%)
```

图 11Multicast 测试 Server 端 5



1

图 12Multicast 测试 Client 端 2

2.1.3 sockperf 3

sockperf 是基于套接字 API 的网络基准测试试用程序，旨在测试高性能系统的性能（延时和吞吐），详细用法可通过 sockperf –h 指令进行了解，此处只介绍部分常见用法。 4

常见用法: 5

表格 5sockperf 的常见用法 6

7

sockperf 常用指令	参数	值	含义
	-h	无	查看使用指南
server<sr>	作为服务端启用		
	-i	<IP>	监听/发送地址
	-p	<Port>	监听/发送端口
	--tcp	无	使用 tcp 协议
	--mc-rx-if	<IP>	接收组播地址
	--mc-tx-if	<IP>	发送组播地址
	--mc-lookback-enable	无	使能本地回环
	--uc-reuseaddr	无	使能端口复用
ping-pong<pp>	作为客户端启用		
	--client_ip	<IP>	绑定作为客户端的地址（用于多网卡多 IP 情况下指定所使用的 IP）
	--client_port	<Port>	绑定作为客户端

			的端口
	--full-log	<FileName>	记录接收信息到 csv 表格中
	上述 server 下指令在此处也可使用		含义与 server 处相同
throughtput<tp>	作为客户端测试吞吐率		
	上述 ping-pong 下指令在此处也可使用		含义与 ping-pong 处相同

1

示例：2

按照示例指令在两个节点间分别启用客户端和服务端，可进行相应模式下的连通性测试。3

表格 6sockperf 的使用示例 4

5

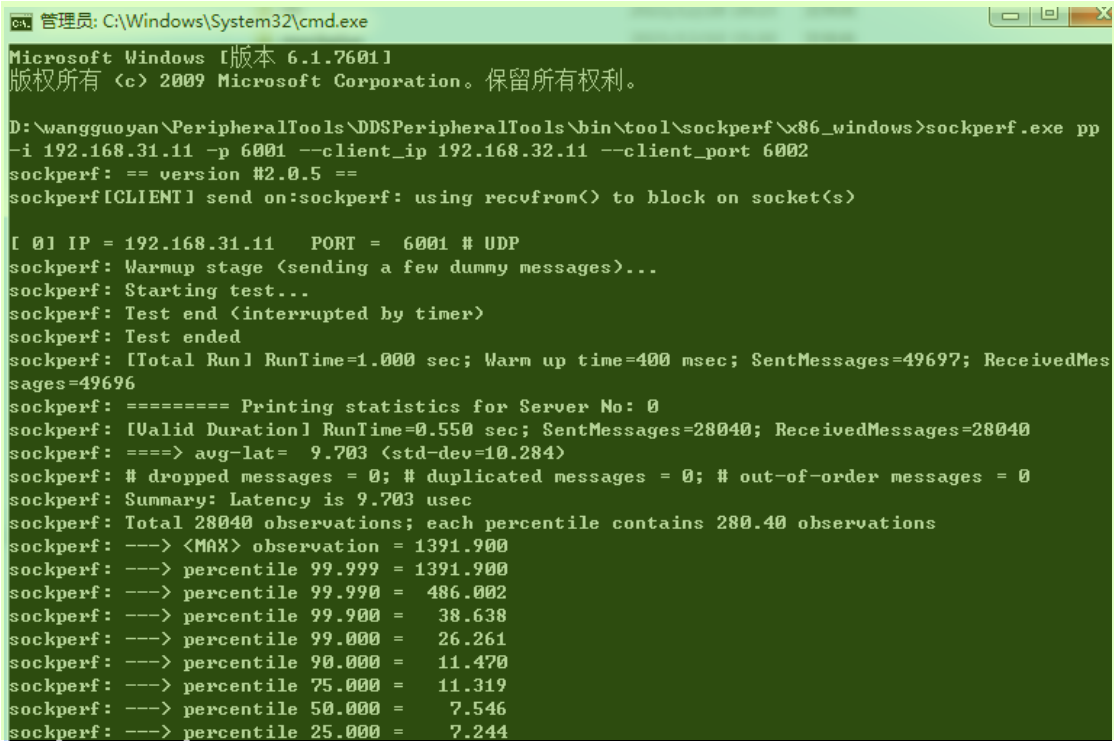
sockperf 收发指令示例		
模式	Client	Server
udp	sockperf.exe pp –i<serverIP> -p <serverPort> --client_ip<clientIP> --clientPort<clientPort> --uc-reuseaddr --full-log LocalUDPUnicast.csv	sockperf.exe server -i <serverIP> –p <serverPort>
tcp	sockperf.exe pp –i <serverIP> -p <serverPort> --client_ip <clientIP> --clientPort<clientPort> --tcp --uc-reuseaddr --full-log LocalTCPUnicast.csv	sockperf.exe server -i <serverIP> –p <serverPort> --tcp
multicast	sockperf.exe pp –i <multicastIP> -p <multicastPort> --mc-tx-if <clientIP> --clientPort<clientPort> --mc-loopback-enable --full-log LocalUDPMulticast.csv	sockperf.exe server -i <multicastIP> –p <multicastPort> --mc-rx-if <serverIP> --mc-loopback-enable
参数值		
<multicastIP>	组播地址	

<multicastPort>	组播端口
<serverIP>	服务端监听地址
<serverPort>	服务端监听端口
<clientIP>	客户端地址
<clientPort>	客户端端口
LocalUDPUnicast.csv	记录 udp 收发信息的表格文件名
LocalTCPUnicast.csv	记录 tcp 收发信息的表格文件名
LocalUDPMulticast.csv	记录组播收发信息的表格文件名

1

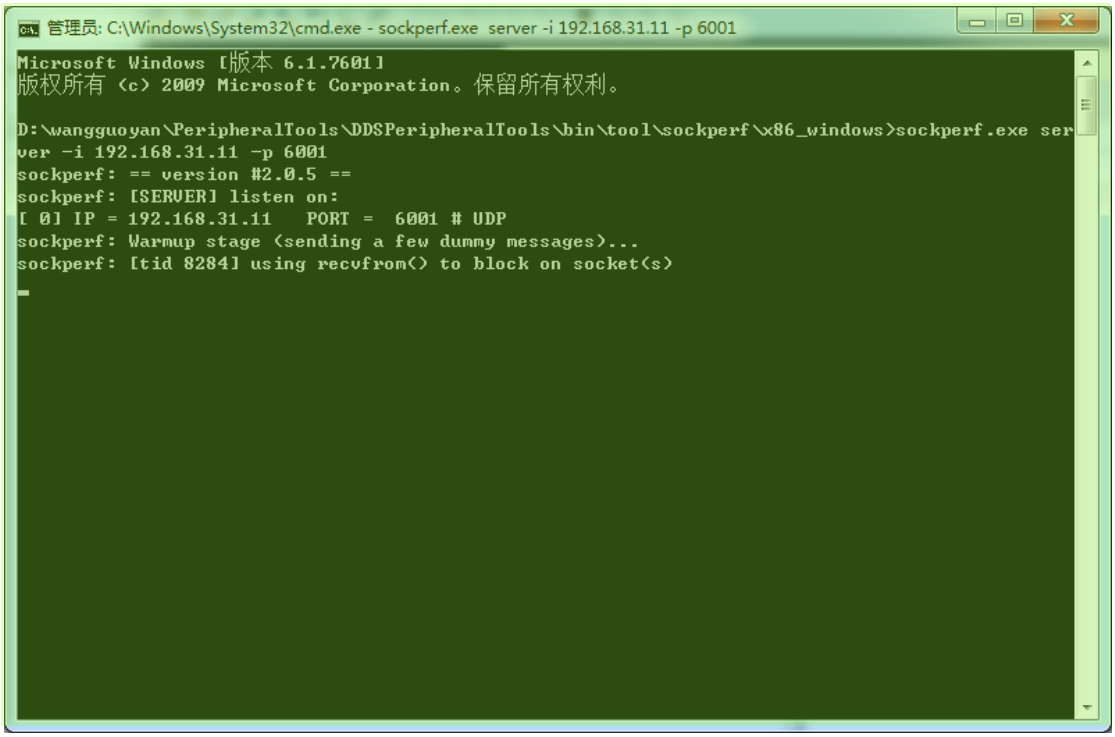
结果示例： 2

UDP 测试：3



4

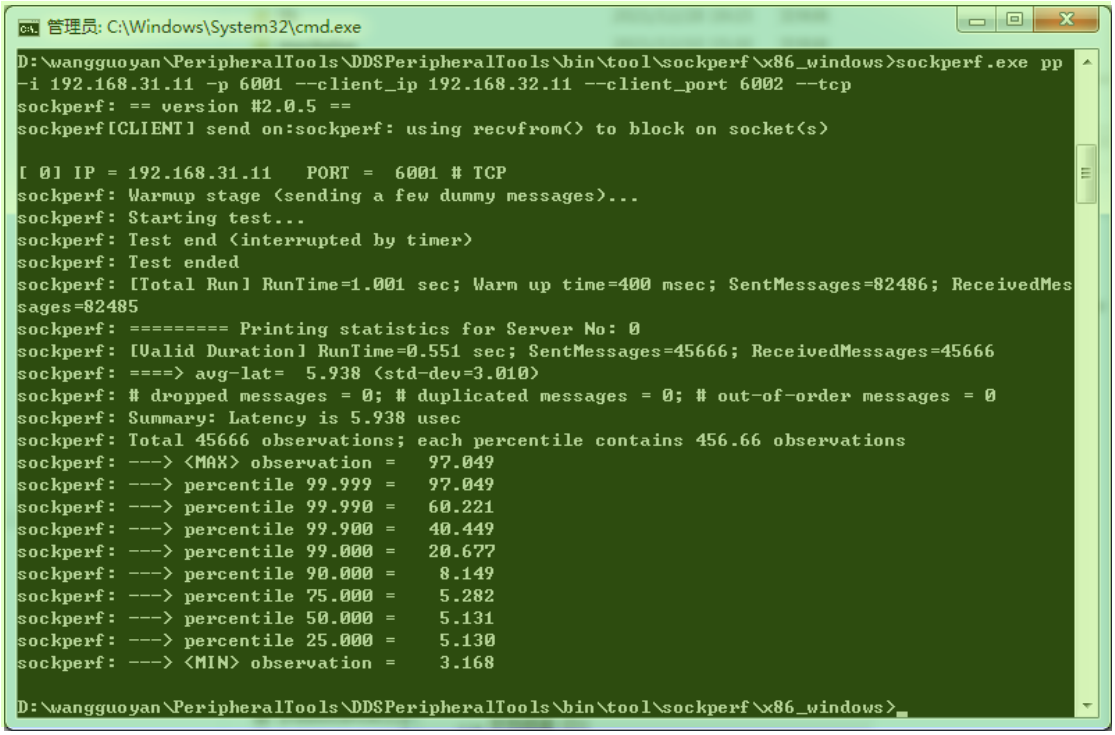
图 13 UDP 测试 Client 端 5



1

图 14UDP 测试 Server 端 2

TCP 测试：3



4

图 15 TCP 测试 Client 端 5

1

```
管理员: C:\Windows\System32\cmd.exe
^C
D:\wangguoyan\PeripheralTools\DDSPeripheralTools\bin\tool\sockperf\x86_windows>sockperf.exe ser
ver -i 192.168.31.11 -p 6001 --tcp
sockperf: == version #2.0.5 ==
sockperf: [SERVER] listen on:
[ 0] IP = 192.168.31.11   PORT =  6001 # TCP
sockperf: Warmup stage (sending a few dummy messages)...
sockperf: [tid 23864] using recvfrom() to block on socket(s)
sockperf: Test end (interrupted by user)
sockperf: Total 67132 messages received and handled
^C
D:\wangguoyan\PeripheralTools\DDSPeripheralTools\bin\tool\sockperf\x86_windows>
```

图 16 TCP 测试 Server 端 2

Multicast 测试：3

4

```
管理员: C:\Windows\System32\cmd.exe
D:\wangguoyan\PeripheralTools\DDSPeripheralTools\bin\tool\sockperf\x86_windows>sockperf.exe pp
-i 224.0.0.1 -p 6001 --mc-tx-if 192.168.32.11 --client_port 6002 --mc-loopback-enable
sockperf: == version #2.0.5 ==
sockperf[CLIENT] send on:sockperf: using recvfrom() to block on socket(s)

[ 0] IP = 224.0.0.1       PORT =  6001 # UDP
sockperf: Warmup stage (sending a few dummy messages)...
sockperf: Starting test...
sockperf: Test end (interrupted by timer)
sockperf: Test ended
sockperf: [Total Run] RunTime=1.003 sec; Warm up time=400 msec; SentMessages=47324; ReceivedMes
sages=47323
sockperf: ========= Printing statistics for Server No: 0
sockperf: [Valid Duration] RunTime=0.553 sec; SentMessages=27207; ReceivedMessages=27207
sockperf: ====> avg-lat= 10.055 (std-dev=2.546)
sockperf: # dropped messages = 0; # duplicated messages = 0; # out-of-order messages = 0
sockperf: Summary: Latency is 10.055 usec
sockperf: Total 27207 observations; each percentile contains 272.07 observations
sockperf: ---> <MAX> observation =   44.071
sockperf: ---> percentile 99.999 =   44.071
sockperf: ---> percentile 99.990 =   39.392
sockperf: ---> percentile 99.900 =   29.129
sockperf: ---> percentile 99.000 =   20.073
sockperf: ---> percentile 90.000 =   12.828
sockperf: ---> percentile 75.000 =   11.017
sockperf: ---> percentile 50.000 =    9.507
sockperf: ---> percentile 25.000 =    8.451
sockperf: ---> <MIN> observation =    5.281

D:\wangguoyan\PeripheralTools\DDSPeripheralTools\bin\tool\sockperf\x86_windows>
```

图 17 Multicast 测试 Client 端 5

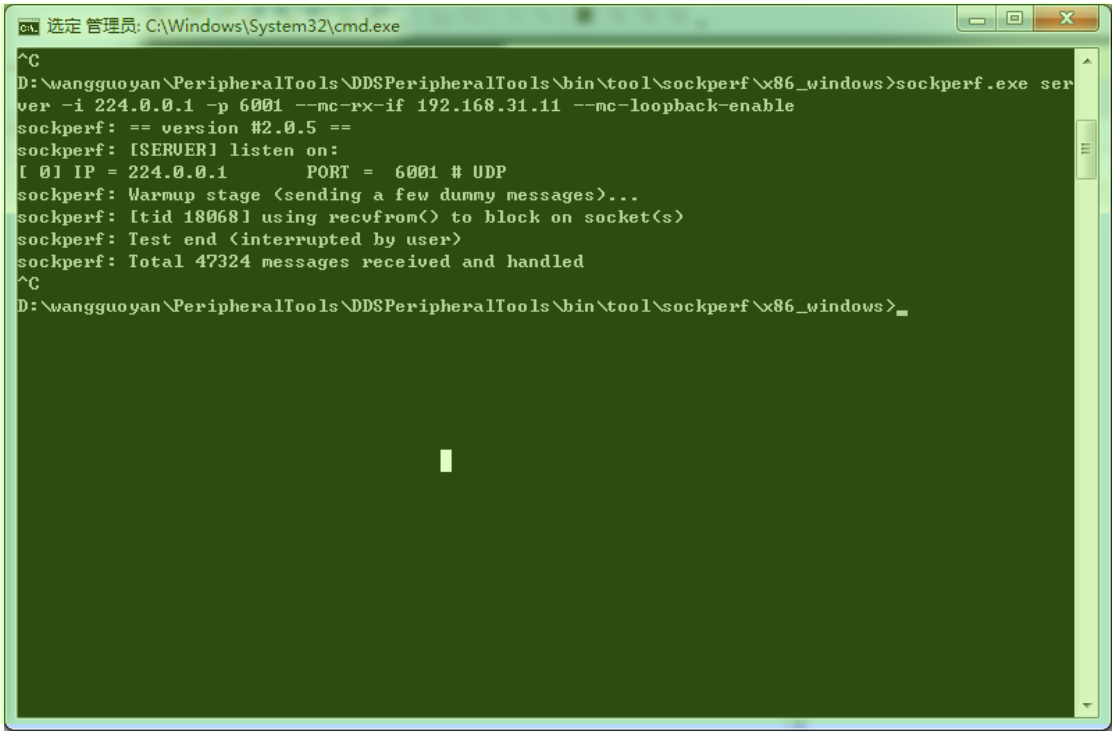


图 18Multicast 测试 Server 端

2.2 交互式日志

在 ZRDDS 的运行过程中，当某些异常或特定条件被触发时，可能会在控制台、日志文件中输出日志信息。这些日志信息一般是错误、警告内容，用于告知在系统运行过程中发生的错误，是最直接的排错手段之一。但是，当系统已经开始输出错误、警告等级的日志，往往是已经在异常发生之后，为时已晚。而仅通过单一的错误、警告日志信息，我们只能拿到发生异常的直接原因，难以推断系统内部的逻辑错误。

为了能够通过日志对系统行为进行深度分析，我们在 ZRDDS 各模块的不同运行阶段的关键点上增加了若干可动态开关的日志信息。即交互式调试日志。我们为不同日志分配了对应的日志号，可以通过控制文件选择性地对交互式调试日志进行开关。

在本小节中，我们将对 ZRDDS 的日志系统、交互式日志调试文件进行介绍。

2.2.1 ZRDDS 日志系统

2.2.1.1 日志级别

在 ZRDDS 中存在四种级别的日志级别，包括错误、警告、系统信息及用户信息。其参数、含义及控制台中日志的颜色参见下表。

表 2.2-1 级别参数与含义对照表

级别	Level	含义	控制台颜色
ZRLOG_ERROR	2	程序出现不可忽略的错误。	红
ZRLOG_ADMIN_INFO	4	系统管理员视角的信息,如一些调试信息。	绿
ZRLOG_USER_INFO	8	通知用户视角信息，即只能看到自定义 Endpoint 之间的数据交互。	默认
ZRLOG_WARNING	16	警告信息但是能够正常运行。	黄

1

2.2.1.2 日志输出通道 2

DDS 日志可以输出到以下三个通道：3

1） 控制台 4

默认情况日志会输出至控制台中。通过控制台查看日志时，可通过日志颜色判断日志等级，控制台中红色代表 ZRLOG_ERROR 级别的错误日志，黄色的代表 ZRLOG_WARNING 级别的警告日志，绿色代表 ZRLOG_ADMIN_INFO 级别的调试日志，以及默认颜色的 ZRLOG_USER_INFO 级别的用户信息日志。5

2） 日志文件 6

默认在程序运行目录中生成名为“应用程序名称.ddslog”的日志文件。如果不存在则创建，如果存在则重写。故而用户在发生故障时，建议在重新启动程序将日志文件进行备份，避免被覆盖。7

2.2.1.3 日志输出控制 8

DDS 日志输出控制示例如下。9

2.2.1.3.1 关闭日志文件输出 10

配置 dpf 的 dds_log 的 qos；dds 只会打印在界面上，而不会输出到文件里 11

```
<participantfactory_qos>
<dds_log>
<file_mask>0</file_mask>
</dds_log>
</participantfactory_qos>
```

2.2.1.3.2 关闭控制台打印 12

关闭终端打印日志的 dpf_qos 配置；13

<participantfactory_qos> 14

<dds_log> 15

```
<console_mask>0</console_mask>
</dds_log>
</participantfactory_qos>
```

2.2.1.4 日志输出格式 1

日志的输出格式如下图所示：2

```
*************************时间      *************************
LEVEL：日志级别
FILE：源码文件名
FUNC：函数名（LINE：源码文件中日志行号)
具体日志内容
*****************************************************************
```

3

例如：4

5

通过该日志的描述结合其文件、行号，可快速定位异常位置。6

2.2.2 ZRDDS 日志控制 7

2.2.2.1 交互式日志调试文件 8

交互式日志调试文件，是用来控制日志输出等级、调试日志输出范围、日志输出通道的配置文件。调试文件名称为“zrdds_debug_file.zrdebug”，将其放置于运行目录中即可。 9

调试文件生效的时机包含两个：10

1. 创建域参与者工厂时，会读取调试文件内容。11
2. DDS 周期性检查（100ms）到调试文件修改时，会重新读取文件内容。12

2.2.2.1.1 调试文件编写规范 13

调试文件由多条单行的调试命令组成，调试命令的格式为：debug_cmd 命令编号命令参数，其中调试命令编号及其参数参见下表： 14

表格 7 调试命令编号与参数对照表 1

命令编号	命令类型	命令参数	示例
0	使能指定调试编号范围	开始编号结束编号	debug_cmd 0 13 15 使能日志号为 13~15 的调试日志
1	取消使能指定调试编号范围	开始编号结束编号	debug_cmd 1 13 15 取消使能日志号为 13~15 的调试日志
2	使能指定主题调试信息	主题名(无需引号，特殊值将在 1)节中介绍)	debug_cmd 2 example 使能主题名为 example 相关的日志
3	取消使能指定主题调试信息	主题名	debug_cmd 3 example 取消使能主题名为 example 相关的日志
4	使能指定实体调试信息	InstanceHandle 的 16 进制串（当 16 进制串为 0 时，为使能所有实体）	debug_cmd 4 c0a80c0300000bf000000001000100c2 使能指定 InstanceHandle 的实体相关日志
5	取消使能指定实体调试信息	InstanceHandle 的 16 进制串	debug_cmd 5 c0a80c0300000bf000000001000100c2 取消使能指定 InstanceHandle 的实体相关日志
6	设置日志级别	控制台掩码文件掩码分布式日志掩码（掩码值将在下一章节进行介绍）	debug_cmd 6 1E 1E 1E 在控制台中输出所有等级的日志 在日志文件中输出所有等级的日志 在分布式日志中输出所有等级的日志

2

调试文件被 ZRDDS 读取后，会生成一个调试响应文件“zrdds_debug_response_file.zrdebug“。当调试文件编写出现语法错误时，ZRDDS 会将错误信息写入响应文件中。因此，当发现调试文件未起效时，可以打开响应文件检查是否存在异常。 3

另外，请注意文件的行尾格式。在 UNIX 上可能会因为调试文件的行尾格式为 Dos/Windows，导致调试文件中的命令无效。 4

2.2.2.1.2 参数介绍 5

1) 特殊主题名称 6

表格 8 特殊主题与含义对照表 7

主题名称	含义
ZRDDS_NETWORK_TOPIC	网络相关操作
ZRDDS_DOMAINPARTICIPANT_TOPIC	域参与者相关调试信息
ZRDDS_DOMAINPARTICIPANTFACTORY_TOPIC	域参与者工厂调试信息
ZRDDS_MPORT_TOPIC	MPORT 实现的 RIO 调试信息
ZRDDS_VALIDATOR_TOPIC	License 验证相关调试信息

8

ZRDDS_INTERACTIVECMD_TOPIC	交互式命令相关调试信息
ZRDDS_ANY_LOG_TOPIC_NAME___	任意主题的调试信息（若开启此名称调试信息，调试文件中关闭某一特定主题的指令将会失效）

1

注：ZRDDS_ANY_LOG_TOPIC_NAME___末尾为 3 个下划线 2

2) 日志级别掩码值 3

设置掩码格式为 debug_cmd 6 [控制台掩码][文件掩码] [分布式日志掩码]。在对应掩码位置输入以下数值，可调整其日志输出等级。此处必须填入十六进制整形。 4

例如 debug_cmd 6 1E 1E 1E，表示在控制台、日志文件、分布式日志中均输出所有等级日志。 5

表格 9 使能级别与参数值对照表 6

使能级别	参数值
不输出任何日志	0
仅输出错误日志	2
输出错误日志、警告日志	12
输出错误日志、警告日志、管理员等级日志	16
输出所有日志	1E

7

2.2.2.1.3 调试文件示例 8

```
//开启日志号为 3002 到 4003 区间内的日志（包括 3002 与 4003）
debug_cmd 0 3002 4003
//取消使能日志号为 3020 到 3030 区间内的日志（包括 3020 与 3030）
debug_cmd 1 3020 3030
//使能域参与者工厂调试主题相关调试信息
debug_cmd 2 ZRDDS_DOMAINPARTICIPANT_TOPIC
//取消使能域参与者相关调试信息
debug_cmd 3 ZRDDS_DOMAINPARTICIPANT_TOPIC
//使能指定实体
debug_cmd 4 00000000000000000000000000000000
//使能级别掩码为允许控制台，文件，分布式日志都使能所有等级的日志
debug_cmd 6 1E 1E 1E
```

9

注意：使能后如果想要关闭对应日志输出，需要取消使能，才能正确地将日志关闭。10

例如，在 ZRDDS 运行过程中，我们想要查看第 13 号日志的相关内容，可以在文件中输入“debug_cmd 0 13 13”并保存。当 ZRDDS 识别到调试文件发生修改，便会做出响应，开始输出第 13 号日志的相关内容。此时，如果我们想要关闭 13 号日志的输出，将 debug_cmd 0 13 13 进行单行注释，修改为“//debug_cmd 0 13 13”并保存，是无法正确关闭该日志输出的。正确的做法是，将“debug_cmd 0 13 13”修改为“debug_cmd 1 13 13”，即取消使能第 13 号日志，才能够正确将该日志关闭。 11

2.2.2.2 日志 API 1

当然，除了交互式日志调试文件可以控制以外，用户还可以通过调用全局 API 进行设置，但这往往意味着需要重新进行编译，不属于运行过程进行动态排故的手段，因此，此处不会对 API 接口进行介绍。如需要，可查阅 ZRDDSCoreInterface\ZRUserLog.h 中相关接口。在调用域参与者工厂的单例后调用相关 API 即可。 2

2.2.3 ZRDDS 常用调试日志号 3

表格 10 ZRDDS 常用调试日志号 4

5

阶段	日志说明	日志号		备注
收发地址绑定	创建域参与者时，设置用于接收报文的地址信息	345	local(%s) create UDPRecvLocatorInfo with kind(%d) for locator(%s)	kind(%d) 数据类型 (1:DOMAIN_KIND 4:PDP_KIND 8:META_KIND 16:USER_KIND)
	将接收地址加入组播	322	local(%s) add addr(%s) membership ifAddr(%08x).	
	设置组播发送接口	326	local(%s) create send locator(%s) and set interface(%08x).	
实体匹配	周期性发送 DATA(p)	13	local(%s) send data(p) to pdpAddr(%s).	
	其它节点的 DATA(p)的版本<2.2.5 或地址数<=1，跳过地址自动排序	45	local(%s) ignore sort locator with participant(%s) version(%u %u) locSize(%u)	
	其它节点的 DATA(p)并对地址自动排序时，调换顺序	49	local(%s) swap participant(%s) activeLloc(%u %s) with unAvailLoc(%u %s)	
	收到 DATA(p)，且完成自动地址排序、认为匹配后打印相关信息	52	local(%s) associate with participant(%s) with meta locator(%s) user locator(%s) lease(%d %u)	可以看出地址是否排序正确
	收到报文时，更新其它域参与者的存活状态	59	local(%s) assert remote dp(%s) liveliness at(%d %u) exsit(%d)	
	周期性检查其它节点的存活状态，认为某节点超时失活	61	local(%s) check dp(%s) at(%d %u) lastSeen(%d %u) lease(%d %u) time out construct data(p)	
	周期性检查其它节点的存活状态，认为某节点仍存活	62	local(%s) check dp(%s) at(%d %u) lastSeen(%d %u) not lease(%d %u)	

			and reset duration to(%llu)	
	数据读者收到数据写者信息时，打印其地址	175	local(%s) get writer(%s) unicast(%s) multicast(%s) writerSpecificLocators(%d) useShmem(%u)	数据读者收到数据写者信息时，打印其地址
	数据读者匹配到数据写者	178	local(%s) match writer(%s) result(%d)	数据读者匹配到数据写者
	数据读者与数据写者解除匹配	183	local(%s) disassociate reader(%s) with writer(%s)	数据读者与数据写者解除匹配
	数据写者匹配到数据读者	197	local(%s) match reader(%s) result(%d)	数据写者匹配到数据读者
	数据写者与数据读者解除匹配	196	local(%s) disassociate writer(%s) with reader(%s)	数据写者与数据读者解除匹配
	数据读者同一主题下不存在相同类型的本地数据写者	189	has no matched local(%s) writer localTopic(%p) localWriterIds(%p) localTypeName(%s) to reader(%s)	
	数据写者同一主题下不存在相同类型的本地数据读者	177	local(%s) has no matched topic(%d) or reader(%p) associate with remote writer(%s)	
数据收发	收到报文，打印报文长度及来源端口	275	domainId(%d) dp(%s) received data(%s) bufCount(%u) recvSize(%u) on locator(%s) from(%s)	recvSize(%u)数据长度 from(%s) 来源网络端口
	收到报文，打印报文内容	276	recvSize(%u) detail packet:	会将报文的 HEX 串打印出来，慎用
	数据写者发送心跳的心跳范围	791	New heartbeat(%u) from %d.%u to %d.%u.	
	数据写者收到数据读者的 ACKNACK 消息	1290	Processing acknack from reader %s, acked sn %lld	
	数据写者处理 ACKNACK 消息时，打印数据读者的状态	1291	Processing acknack from reader %s, last ack %lld, last nack %lld.	
	数据读者将数据标为已被接收（ACKED）	773	Informing sample %lld acked.	
	数据读者将数据标为已被发送（提交到网络）后数据写者释放信号量，驱动被阻塞的发送线程	774	Informing sample sending waiting thread on sample %lld.	
	数据读者将数据标为已被接收（ACKED）后数据写者释放信号量，驱动用户 waitForAllAcked 的线程	775	Informing sample acking waiting thread on sample %lld.	
	数据写者在某些数据被 ACKED	776	Trying to remove not alive	

	后释放已经不存活的实例		instance %s.	
	数据写者收到数据读者的NACK_FRAG 消息	1310	Informing nack frag from reader %s, sn %lld.	
	数据写者重发 DATA	1293	Resend sample %lld to reader %s.	
	数据写者重发 DATA_FRAG	1313	Resend frag %lld:%u to reader %s.	
	数据写者发送 GAP	1150	Sending gap from %lld to %lld to reader %s.	
	数据读者收到数据写者的心跳，打印	474	reader(%s) received hbMsg from srcId(%s) reset writer counter(%u) local reliability(%u) historySeq(%d %u) expected(%d %u) fromHeartbeatBatch(%d)	
	数据读者收到过期的数据写者的心跳，忽略	473	reader(%s) ignore hbMsg from srcId(%s) due to counter(%u) leased	
	数据读者处理数据写者的心跳后，打印接收窗口及资源占用状态	479	reader(%s) m_curAvailSampleNum(%d) m_curFreeSampleNum(%d)update writer recvWindow after handle hb(%d.%u - %d.%u)send hb reply isFinal(%d) rightDiff(%lld)	
	数据读者收到数据写者的心跳后，构建 ACKNACK	590	reader(%s) construct acknack for writer(%s) lastSeq(%d %u) maxResource(%u) curAvailSampleNum(%u) dataFragInfo(%u) rejectedByInstance(%d) seqNumSet	
	数据读者收到数据后通知用户（调用用户 listener）	466	self(%s) info user with ret(%d) mask(%u) listener(%p) on_data_available(%p)	
	数据读者样本资源不足以存储数据，并拒绝（reject）	545	reader(%s) loan (%u) sample for dataMsg from srcId(%s) failed.	
	数据读者实例资源不足，拒绝并抛弃（reject&lost）	448	reader(%s) SubscriptionInstanceNew(%s) for sample(%d %u) wrier(%d %u) failed info rejected	
	数据读者为某个样本获取实例下样本空间（max_samples_per_instance）时失败，并拒绝（reject）	454	reader(%s) reject and lost(%d) sample due to purge instance(%s) failed	
	数据读者为实例的第一个样本创建实例空间	455	reader(%s) received new instance(%s) sample create and	

			copy keySample(%p) ret(%d) "	
	数据读者接收到数据后，将数据存到对应数据写者的接收窗口后，打印接收窗口状态	494	reader(%s) m_curAvailSampleNum(%d) m_curFreeSampleNum(%d)\nstore sample(%d %u) from writer to index(%u) offset(%u) update	
	数据读者收到 GAP，打印 GAP 及数据读者自身状态	571	reader(%s) m_curAvailSampleNum(%d) m_curFreeSampleNum(%d) received gapMsg from srcId(%s) leftSeqDiff(%d) rigthSeqDiff(%d)	
	数据读者处理 GAP，并将样本标记为丢失	576	reader(%s) mark sample(%d %u) lost remove frag(%d)	
	数据读者处理 GAP 完成后打印自身状态及接收窗口状态	579	reader(%s) m_curAvailSampleNum(%d) m_curFreeSampleNum(%d)update writer recvWindow after handle gap	
	数据写者尝试使用新样本数据替换内存中的样本空间（keep_last）	671	Trying to obtain sample with size %u. Keep last, history depth %d. Instance samples %u, first sn %lld.	
	数据写者尝试使用新样本数据替换内存中的单个实例的样本空间（keep_all）	673	Trying to obtain sample with size Keep all, max samples per instance %d. Instance samples %u, first sn %lld.	
	数据写者尝试使用新样本数据替换内存中的所有实例的样本空间（keep_all）	675	Trying to obtain sample with size %u. Keep all:%d, max samples %d. Samples %u, first sn %lld.	
	数据写者替换到样本空间	676	New sample with size %u is obtained. Sn %lld, frags %u	
	可靠数据写者由于资源不足进入等待 Acked 的阻塞状态	870	Waiting for sample %lld acked.	
	可靠数据写者由于超时，未成功等到 Acked，解除等待资源的阻塞状态	871	Waiting for sample %lld acked failed.	
	可靠数据写者等到 Acked，解除等待资源的阻塞状态	872	Waiting for sample %lld acked succeeded.	
	数据写者进入 waitForAllAcked 的状态	890	Waiting for all samples acked. Current reliable readers %u	
资源回	数据写者发送某个实例的 UD 消息	1190	Trying send disposed(%d) or unregistered(%d) information for instance %s.	

收	数据写者归还样本空间	730	Returning sample with sn %lld, length %u.	
	数据写者归还 DATA_FRAG 空间	731	Returning sample frag with sn %lld, length %u.	
	存在未删除的域参与者导致资源回收失败	253	factory(%p) has participants(%u) not deleted.	

1

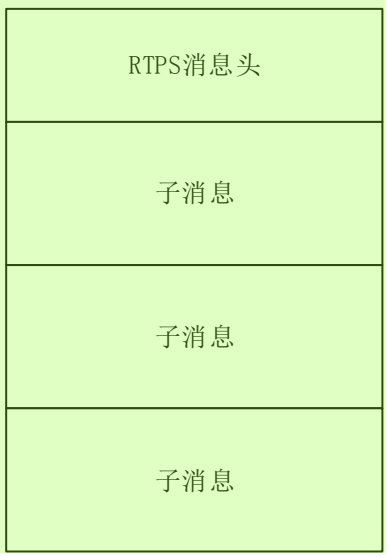
2.3 抓包工具 2

2.3.1 DDS 的数据包封装 3

DDS 的数据包封装格式遵循 OMG RTPS v2.2 标准（兼容 v2.1）。由于协议内容过多，在此处仅对重点部分进行说明。 4

1) RTPS 消息 5

DDS 发出的每一个数据包都是一个 RTPS 消息。RTPS 消息由一个 RTPS 消息头和若干个子消息组成，在结构上如图 19 所示。 6



7

图 19 RTPS 消息结构 8

在 RTPS 消息头中，包含发出该消息的 DomainParticipant GUID 前缀、协议版本号、厂商 ID 号。其后紧接若干个子消息。 9

2) 子消息 10

子消息是 RTPS 中的最小消息单元，所有的 RTPS 消息都包含一个或若干个子消息。子消息包含一个子消息头和子消息内容。如图 20 所示。 11

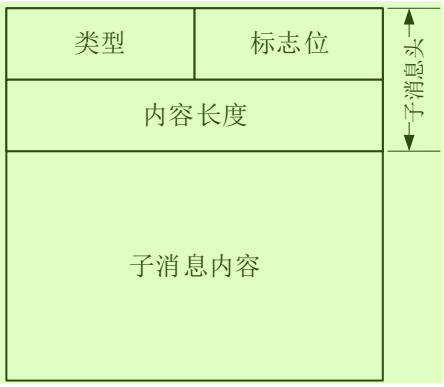


图 20 子消息结构

所有子消息的头部结构都完全一致，而内容部分则根据子消息类型的不同而不同。子消息头部包含子消息类型、标志位和子消息内容长度。在当前 DDS 中支持的子消息类型包含：INFO_TS、INFO_DST、ACKNACK、HEARTBEAT、DATA、GAP、NACK_FRAG、DATA_FRAG。其含义如表 2 所示。

表 2 子消息含义

子消息	长度*	发出者	接收者	含义
INFO_TS	8	不限制	不限制	用于声明消息的时间戳
INFO_DST	12	不限制	不限制	用于声明消息的目的端
ACKNACK	不定	数据读者	数据写者	数据读者表明自身已收到和缺失的数据序号
HEARTBEAT	28	数据写者	数据读者	数据写者表明自身保存的数据序号范围
DATA	不定	数据写者	数据读者	承载数据内容
GAP	不定	数据写者	数据读者	数据写者表明某些序号的数据已经不存在
NACK_FRAG	不定	数据读者	数据写者	数据读者表明自身缺失的分片编号
DATA_FRAG	不定	数据写者	数据读者	承载分片数据内容

*注：仅指内容长度

对于未在表中列出的子消息，可以通过其头部指明的长度跳过，不会影响对后续子消息的解析。

各个子消息的具体结构不再此处详述，可参考 RTPS 协议相关部分。

用户提交的数据通常都是通过 DATA 或 DATA_FRAG 承载，包括 DDS 内置主题的数据也是通过这两个子消息承载。其他子消息是作为状态同步、可靠性保障等功能服务的。对应到本文件第一章第 3 节中的描述，数据对应 DATA 子消息，心跳对应 HEARTBEAT 子消息，反馈对应 ACKNACK 子消息，丢失对应 GAP 子消息。

2.3.2 tcpdump

tcpdump 主要用来抓取 linux 环境下的数据包，需要具备 root 权限的用户才能使用，可以将抓取的数据包保存为 cap 格式，然后使用 WireShark 工具进行分析。

表格 11 tcpdump 常用命令 1

命令	说明
-c	在收到指定的数量的分组后，tcpdump 就会停止
-D	打印出系统中所有可以用 tcpdump 截包的网络接口
-f	将外部的 Internet 地址以数字的形式打印出来
-F	从文件读取表达式，忽略命令行中的表达式
-i	指定网络接口
-w	指定保存文件
-nn	不进行端口名称的转换
-P	不将网络接口设置为混杂模式
-v	输出一个稍微详细的信息，例如在 ip 包中可以包括 ttl 和服务类型的信息

2

示例：抓取网卡 eth0 上的网络数据包，并保存为 out.cap 3

tcpdump –i eth0 –w out.cap 4

2.3.2.1 条件抓包 5

设置固定时长抓一次包（循环抓包）: 6

参数：-G 设置时间（s）7

例如：tcpdump -i ens33 -s0 -G 10 -w %Y_%m%d_%H%M_%S.pcap 8

设置报文到达指定数量保存; 9

参数：-c 10 设置包数 ； -s0：指定大小，（如果不加这个参数，超出 10MB 的部分将被丢弃） 10

例如：tcpdump -i ens33 -s0 -c 10 -w a.pcap 11

设置每 N 兆（1000000 字节）抓一次 12

参数：-c N 设置大小 13

例如： tcpdump -i ens33 -s0 -c 1 -w a.pcap 14

抓组播的包 15

例如：Tcpdump – eth0 ‘ip multicast and udp and dst 239.255.0.1 and src haost [本机 ip]’; 16

2.3.3 WireShark 17

Wireshark 是常用的网络抓包和分析工具。18

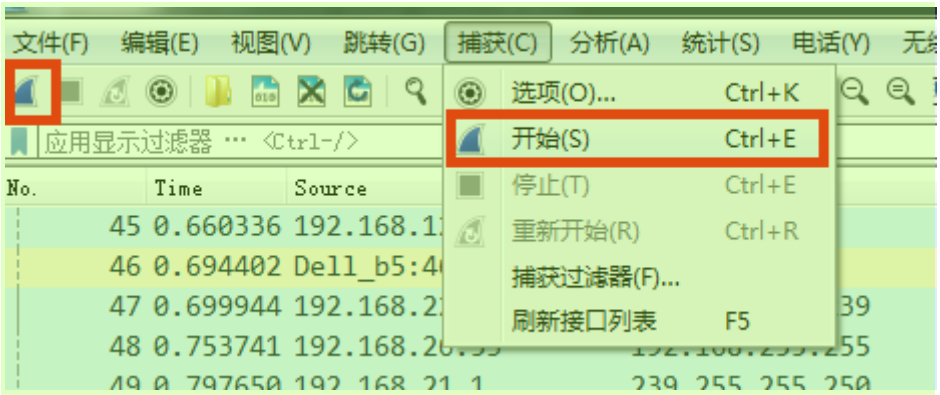
2.3.3.1 WireShark 数据抓包 1

a) 选择本地网卡 2

3

图 21 选择抓取网卡 4

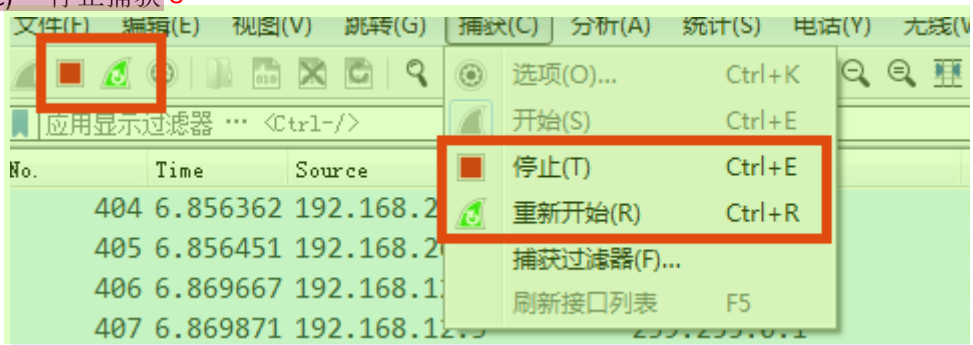
b) 捕获数据 5



6

图 22 开始捕获 7

c) 停止捕获 8



9

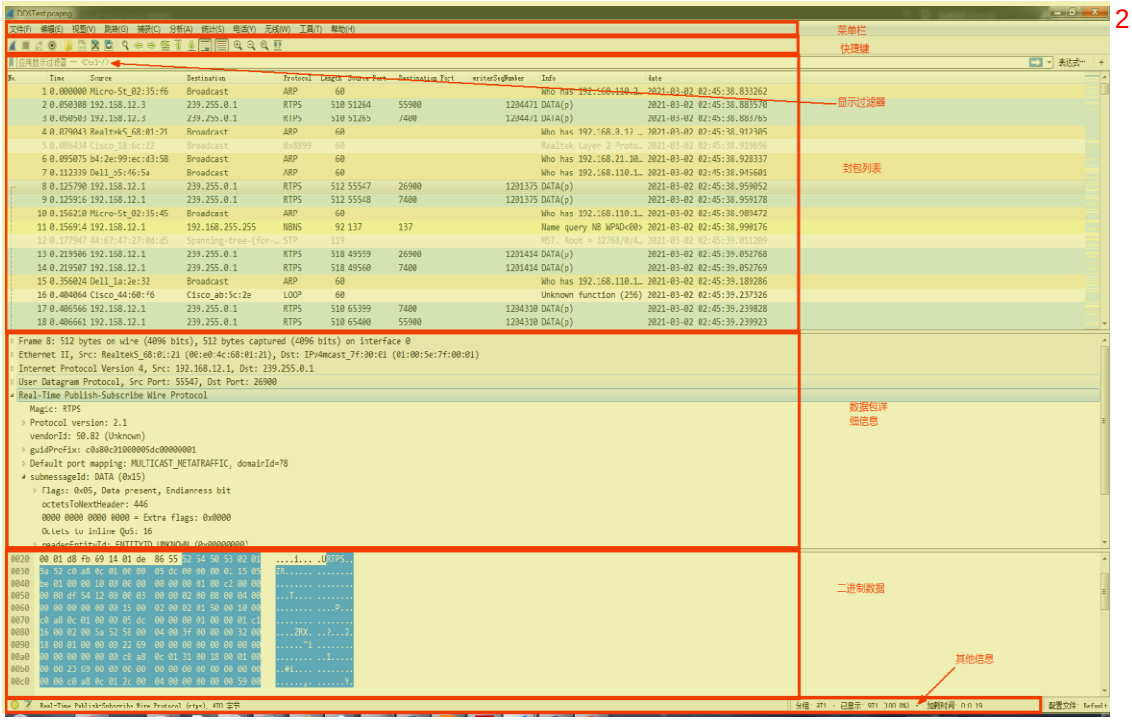
图 23 停止捕获 10

d) WireShark 可以设置抓取的数据包数量和大小，可以筛选数据协议。11

2.3.3.2 WireShark 抓包分析 12

在 Wireshark 中，内置了 DDS 数据包的解析器，可以帮助用户分析 DDS 的通信过程，排除通信故障。 13

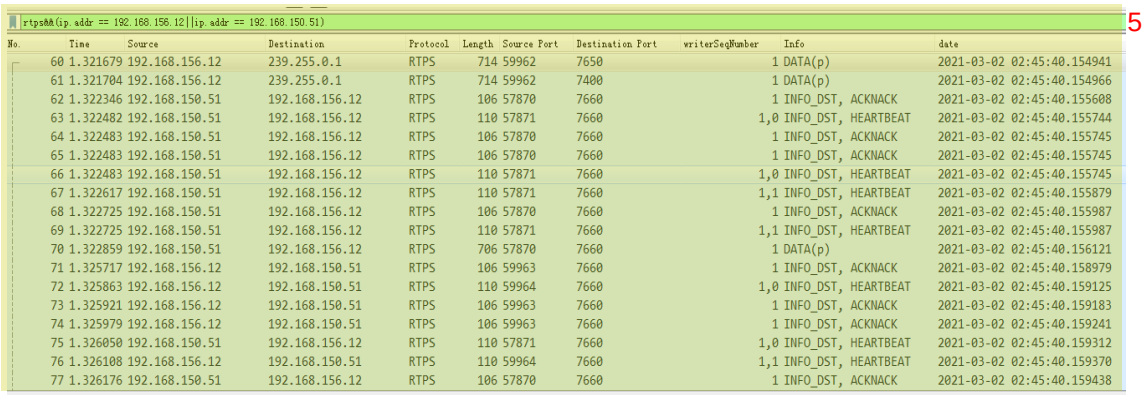
一下为 WireShark 数据分析界面：1



2

图 24 数据分析图 3

数据过滤：可以通过表达式过滤出自己感兴趣的数据 4



5

No.	Time	Source	Destination	Protocol	Length	Source Port	Destination Port	writerSeqNumber	Info	date
60	1.321679	192.168.156.12	239.255.0.1	RTPS	714	59962	7650	1	DATA(p)	2021-03-02 02:45:40.154941
61	1.321704	192.168.156.12	239.255.0.1	RTPS	714	59962	7400	1	DATA(p)	2021-03-02 02:45:40.154966
62	1.322346	192.168.150.51	192.168.156.12	RTPS	106	57870	7660	1	INFO_DST, ACKNACK	2021-03-02 02:45:40.155608
63	1.322482	192.168.150.51	192.168.156.12	RTPS	110	57871	7660	1,0	INFO_DST, HEARTBEAT	2021-03-02 02:45:40.155744
64	1.322483	192.168.150.51	192.168.156.12	RTPS	106	57870	7660	1	INFO_DST, ACKNACK	2021-03-02 02:45:40.155745
65	1.322483	192.168.150.51	192.168.156.12	RTPS	106	57870	7660	1	INFO_DST, ACKNACK	2021-03-02 02:45:40.155745
66	1.322483	192.168.150.51	192.168.156.12	RTPS	110	57871	7660	1,0	INFO_DST, HEARTBEAT	2021-03-02 02:45:40.155745
67	1.322617	192.168.150.51	192.168.156.12	RTPS	110	57871	7660	1,1	INFO_DST, HEARTBEAT	2021-03-02 02:45:40.155879
68	1.322725	192.168.150.51	192.168.156.12	RTPS	106	57870	7660	1	INFO_DST, ACKNACK	2021-03-02 02:45:40.155987
69	1.322725	192.168.150.51	192.168.156.12	RTPS	110	57871	7660	1,1	INFO_DST, HEARTBEAT	2021-03-02 02:45:40.155987
70	1.322859	192.168.150.51	192.168.156.12	RTPS	706	57870	7660	1	DATA(p)	2021-03-02 02:45:40.156121
71	1.325717	192.168.156.12	192.168.150.51	RTPS	106	59963	7660	1	INFO_DST, ACKNACK	2021-03-02 02:45:40.158979
72	1.325863	192.168.156.12	192.168.150.51	RTPS	110	59964	7660	1,0	INFO_DST, HEARTBEAT	2021-03-02 02:45:40.159125
73	1.325921	192.168.156.12	192.168.150.51	RTPS	106	59963	7660	1	INFO_DST, ACKNACK	2021-03-02 02:45:40.159183
74	1.325979	192.168.156.12	192.168.150.51	RTPS	106	59963	7660	1	INFO_DST, ACKNACK	2021-03-02 02:45:40.159241
75	1.326050	192.168.150.51	192.168.156.12	RTPS	110	57871	7660	1,0	INFO_DST, HEARTBEAT	2021-03-02 02:45:40.159312
76	1.326108	192.168.156.12	192.168.150.51	RTPS	110	59964	7660	1,1	INFO_DST, HEARTBEAT	2021-03-02 02:45:40.159370
77	1.326176	192.168.150.51	192.168.156.12	RTPS	106	57870	7660	1	INFO_DST, ACKNACK	2021-03-02 02:45:40.159438

图 25 过滤后数据 6

分析：7

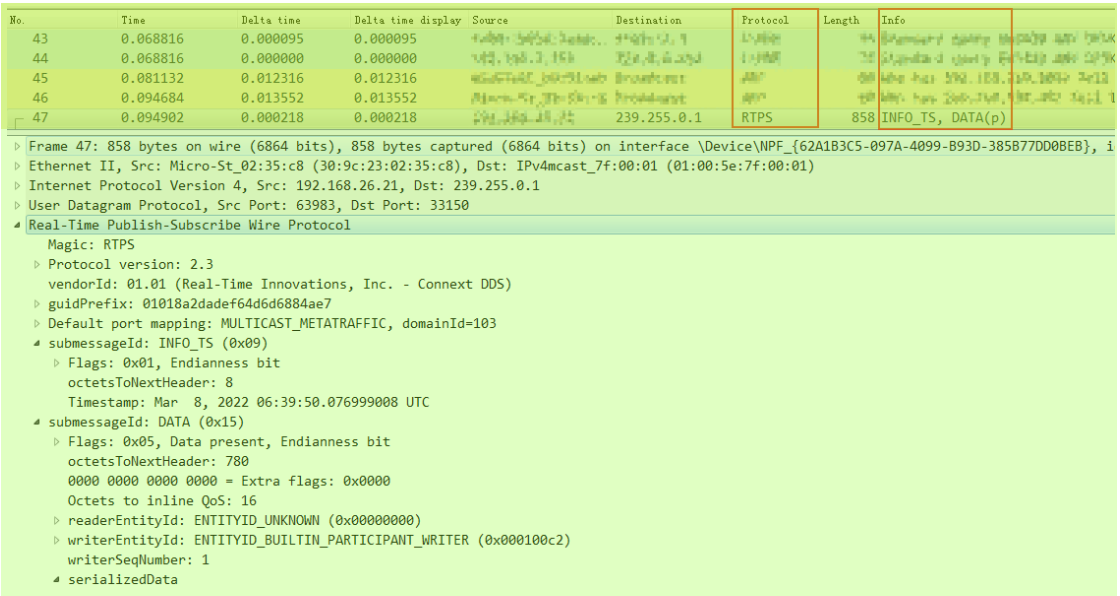


图 26 Wireshark 解析 DDS 数据包

如图 26 所示，通过 Wireshark 抓取的一个 DDS 的数据包可以直接被解析出来，可以看到图中两个红框，左边红框里是数据包的协议，已经被解析为 RTPS，右边红框里的数据包信息显示了该数据包中包含两个子消息，分别为 INFO_TS 和 DATA。在下面的详细信息中，数据包内的每个字段都以可读的形式显示，能大大提升数据分析效率。在 Wireshark 的过滤器中填入“rtps”（不包含引号），可以将其他消息过滤，只留下 RTPS 消息。

在图 26 中，注意到其中显示的 DATA 子消息后还包含一个“(p)”后缀。这一后缀号表示这是一个属于“Participant”的 DATA 子消息。通常这类子消息就是 DDS 用于发现的数据包。Wireshark 还能识别出保存了 DataWriter 匹配信息的 DATA(w)子消息，保存了 DataReader 匹配信息的 DATA(r)子消息。而用户的数据，通常仅标识为“DATA”而不包含任何后缀。

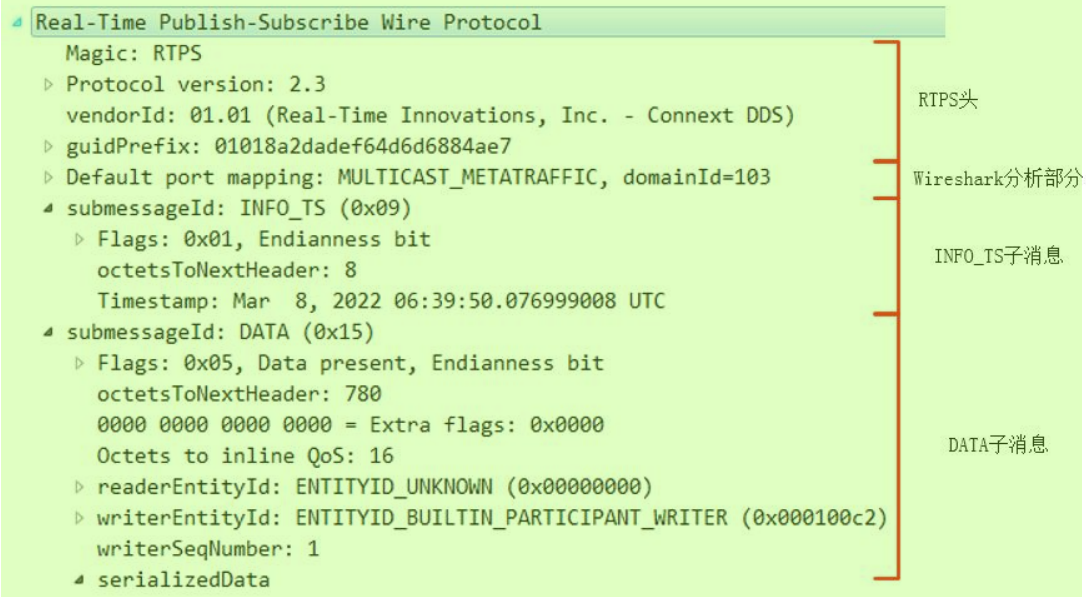


图 27 RTPS 消息详情

在 RTPS 详情中，我们可以清晰地看到 RTPS 的消息结构，如图 27 所示。消息起始部分是 RTPS 消息头，后面还包含了 Wireshark 分析得到的域号。紧接着是一个 INFO_TS 子消息，

包含了时间戳的信息。而后是一个 DATA 子消息，包含了数据内容。用户还可以展开各个子项查看更详细的信息。

表格 12 子消息分析常用字段

子消息	字段	说明
RTPS 消息	guidPrefix	通过该字段是 DomainParticipant GUID 的前 12 个字节，可以确定消息所属的 DomainParticipant，因为 DomainParticipant 的 GUID 后 4 个字节都是固定的，仅使用前 12 个字节区分
	domainId	该字段表示消息来自于哪个域号，在用户没有设定接收地址时该字段分析是准确的，如果用户设定了接收地址和端口，该字段则不能作为参考
【通用】	octetsToNextHeader	当前子消息内容的长度，通常可用于识别或过滤特定长度的数据
DATA	writerSeqNumber	数据的序列号，表明数据是当前 DataWriter 发出的第几个数据
	writerEntityId	发出该数据的 DataWriter 标识，Wireshark 可分析出是由内置主题还是用户主题的的 DataWriter 发出，可以初步确定数据归属的主题，如果结合 DATA(w) 数据中的 Guid 信息，可以确定数据所属主题
	readerEntityId	数据发往的 DataReader 表示，如果是 ENTITYID_UNKNOWN 表示发送给所有同主题的 DataReader
	serializedData	用户的数据内容，在知晓主题的数据结构和数据序列化规则之后，可以通过该字段手动解析出数据内容，特定场景下能有效帮助定位问题
DATA_FRAG	writerSeqNumber	当前分片数据所属的完整数据的序列号
	fragmentStartingNum	当前分片数据的首个分片号，通常一个数据包仅包含一个分片数据，因此该值就是分片数据的分片号
HEARTBEAT	firstAvailableSeqNumber	**未被 readerEntityId 指定 DataReader 确认**的 DataWriter 数据队列中的第一个数据序列号，如果当前子消息的 readerEntityId 是 ENTITYID_UNKNOWN，则表示**未被所有 DataReader 确认**的数据队列中的第一个数据序列号
	lastSeqNumber	DataWriter 数据队列中最后一个数据的序列号，如果该值小于

		firstAvailableSeqNumber，则表明 readerEntityId 指定的 DataReader（若 readerEntityId 是 ENTITYID_UNKNOWN 则表示所有 DataReader）已经确认了当前 DataWriter 发出的所有数据
GAP	gapList	DataWriter 声明的缺失数据的序列号，较新版本的 Wireshark 可以直接显示出序列号的列表
ACKNACK	readerSNState	DataReader 声明的已经收到和缺失的数据序列号，较新版本的 Wireshark 可以直接显示出序列号的列表
NACK_FRAG	wirterSN	分片所属完整数据包的序列号
	fragmentNumberState	缺失的分片信息

上表列出了使用 Wireshark 分析 RTPS 子消息时常用的字段及含义。

需要注意的是，Wireshark 抓包会占用大量内存和磁盘空间，如果应用程序正在大量收发数据，有可能导致内存被快速占满，且 CPU 被抢占，影响其他应用程序甚至系统的运行。同时，如果用户使用版本较低的 Wireshark 抓包时，无法抓取到本地回环的数据包，有可能产生应用程序明明在正常通信但是 Wireshark 上看不到任何数据包的情况。如果在 Windows 平台可以使用 RawCap 工具抓取本地回环的数据包，并将其保存的文件通过 Wireshark 进行解析。

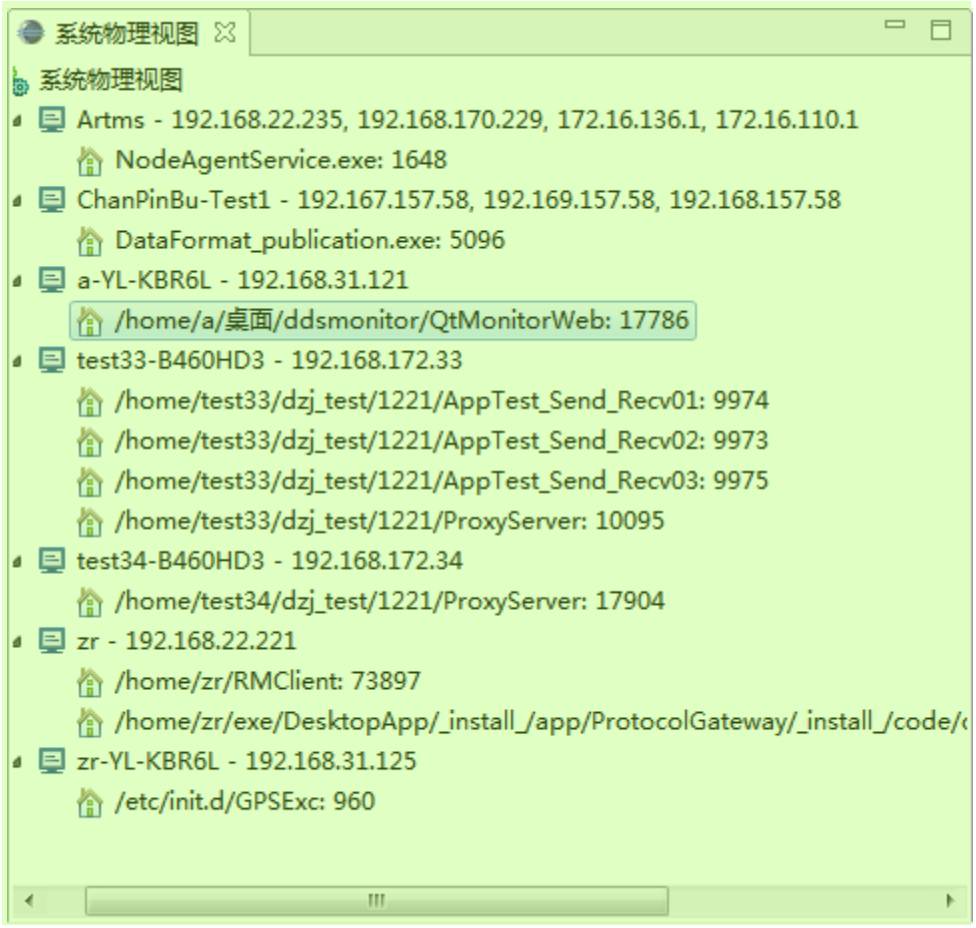
除了使用 Wireshark 抓包之外，Linux 平台上还可以使用 tcpdump 进行抓包。如果保存成文件，同样可以使用 Wireshark 打开并解析。

2.4 ZRDDS 管理监控器

第一步：检查域是否相同以及主题名是否一致

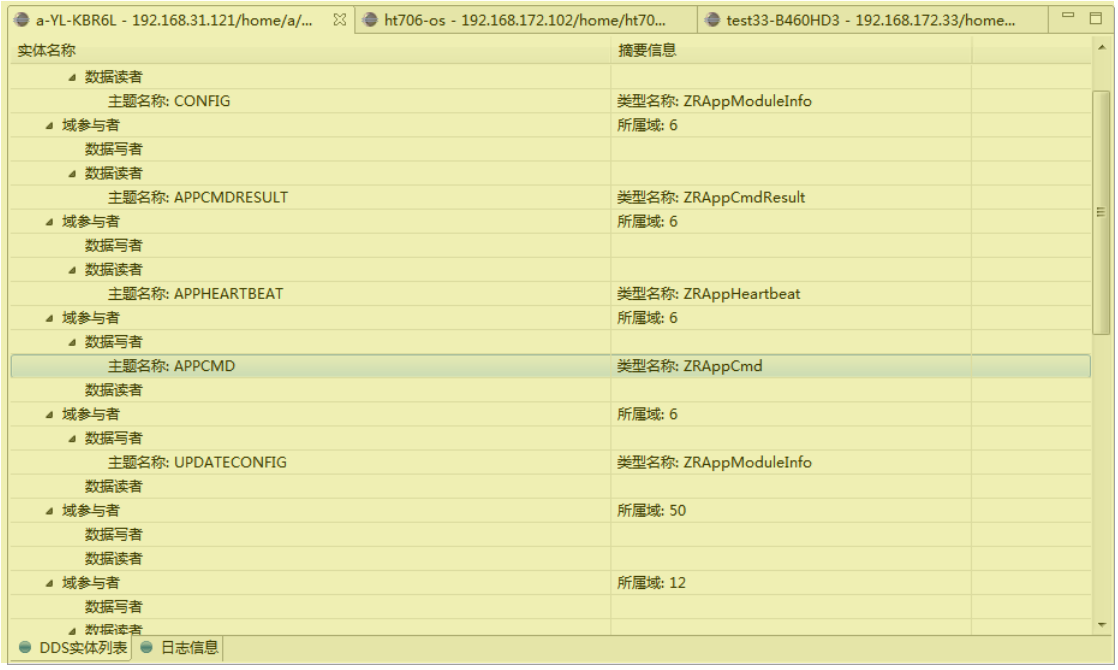
在确保物理网络正常的情况下，双方无法进行通信，需要先检查双方的域号以及主题名是否一致。

如图 28 所示，在“系统物理视图”中，找到通信双方所属的进程。双击显示进程名，在如图 29 所示，在进程详细信息视图中，查看实体（数据读者/数据写者）所属的域，以及主题名。对比通信双方是否在相同的域，以及主题名和类型名是否相同。



1

图 28 系统物理视图 2



3

图 29 进程详细信息 4

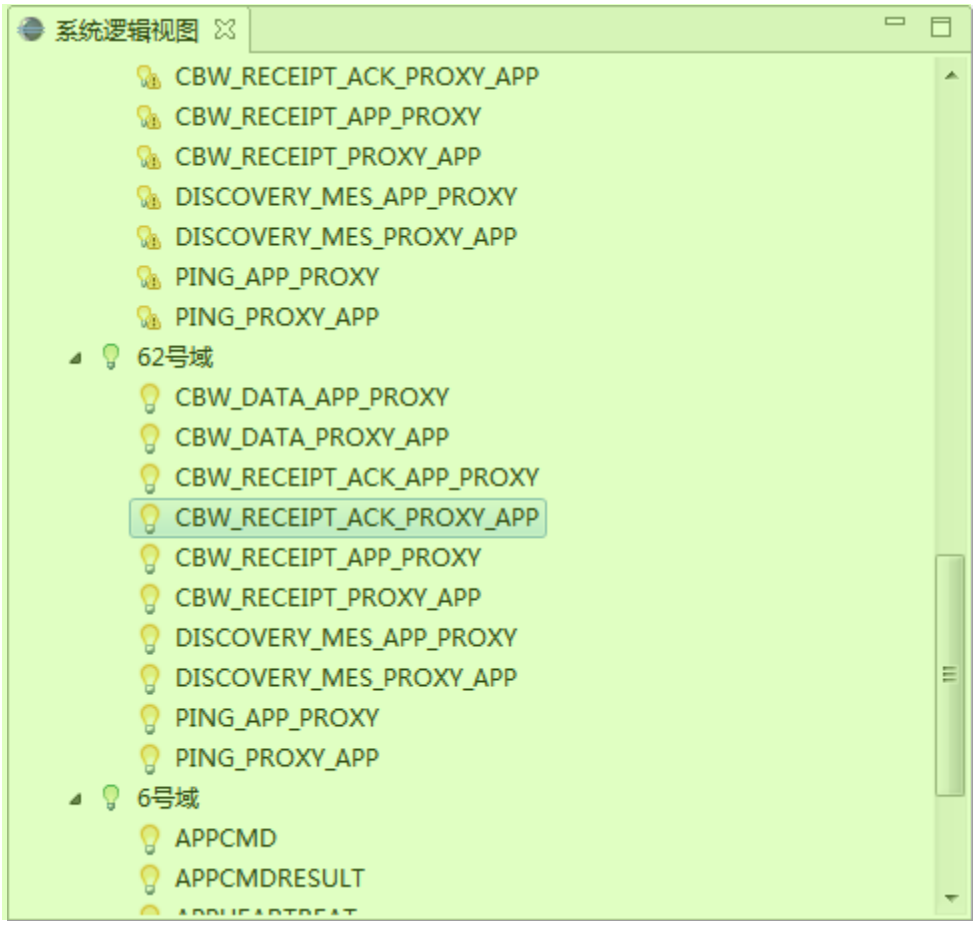
第二步：数据类型以及 QoS 是否相同 1

若能够确定通信双方的域号、主题名和数据类型名相同，但通信双方仍旧无法通信，则需要进一步判断通信是否能够匹配上（请注意：这里使用的是“能够”，因为监控器是通过监控到的数据读者和数据写者的信息，来模拟双方是否应该匹配上）。 2

如图 31 所示，在“系统逻辑视图“ ”中，找到需要检测的实体所属的域号和主题名并双击。弹出如图 31 所示的“主题详细信息”视图，在“实体列表”的标签页中，可以看到该域号该主题下数据读者和数据写者的信息。通过状态信息，可以初步判断双方是否匹配上了： 3

- Normal：代表数据读者和数据写者能够匹配； 4
- warning：代表该域号该主题下，仅有数据读者或者数据写者； 5
- error：代表数据读者和数据写者不能够匹配。 6

若是不能够匹配，则需要进一步查看是哪些数据读者和数据写者没能匹配上，以及没有匹配上的原因。在“实体列表”标签页中，选中一实体，则在如图 32 所示的视图中则显示其需要匹配的实体信息，并显示是否匹配上以及不匹配的原因。（加入选中的实体时数据写者，则“匹配分析”视图中显示的则是“实体列表”标签页中数据读者的信息）。 7



8

图 30 系统逻辑视图 9

62号域：CBW_RECEIPT_ACK_PROXY_APP主题详细信息 1

实体类型	节点	进程/模块名	进程/任务号	标识	状态
DataWriter	test33-B460HD3 - 192.168.172...	/home/test33/dzj_test/1221/Pr...	10095	0xc0a8ac210000276f0000000016000...	Normal
DataReader	test33-B460HD3 - 192.168.172...	/home/test33/dzj_test/1221/A...	9975	0xc0a8ac21000026f70000000019000...	Normal
DataReader	test33-B460HD3 - 192.168.172...	/home/test33/dzj_test/1221/A...	9973	0xc0a8ac21000026f50000000019000...	Normal
DataReader	test33-B460HD3 - 192.168.172...	/home/test33/dzj_test/1221/A...	9974	0xc0a8ac21000026f60000000019000...	Normal

实体列表 | 主题匹配图 | 主题数据订阅 | 主题数据发布

图 31 主题详细信息视图 2

匹配分析 3

检查项	匹配规则	发布端	订阅端
0xc0a8ac21000026f7000000001900000	/home/test33/dzj_test/1221/AppTest_Send_Recv03: 9975		是
类型名称	类型名称必须相等	FileReceipt	FileReceipt
类型编码	类型编码必须相同	struct ::InsCon...	struct ::InsContent(sequen...
QoS匹配项	QoS必须满足RxO规则		
durability			
presentation	需要满足以下三个条件		
deadline			
latency_budget			
ownership			
liveliness	需满足以下两个条件		
reliability			
destination			
0xc0a8ac21000026f5000000001900000	/home/test33/dzj_test/1221/AppTest_Send_Recv02: 9973		是
0xc0a8ac21000026f6000000001900000	/home/test33/dzj_test/1221/AppTest_Send_Recv01: 9974		是

图 32 匹配分析视图 4

注：ZRDDS 管理监控器的介绍以及使用方法，参照《ZRDDS 管理监控器使用手册》5

3 典型故障 1

3.1 编译失败 2

3.1.1 环境变量未生效 3

可能存在在工程中通过 ZRDDS_HOME 环境变量链接库和头文件，然而实际情况下 ZRDDS_HOME 环境变量未生效，导致编译失败。4

故障表现： 5

编译时报错库文件，头文件找不到。6

排查方式： 7

若工程所链库和头文件地址是基于 ZRDDS_HOME 环境变量，则检查环境变量是否生效。8

Window 下在 cmd 窗口中输入 set 指令，查看环境变量中是否存在 ZRDDS_HOME 且值是否正确，在 Linux 下输入 echo $ZRDDS_HOME 查看。9

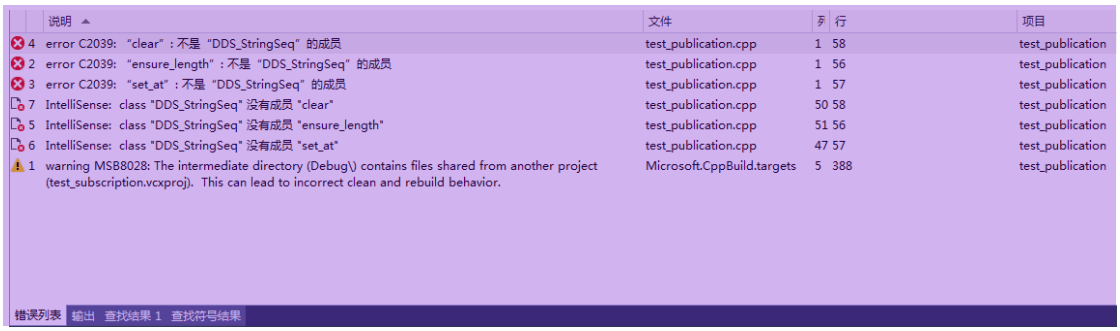
通常在设置环境变量后需重启才可使环境变量生效，若发现环境变量已写入但未生效可通过重启的方式使其生效。10

3.1.2 预编译符缺失 11

可能存在 C++工程中未填写预编译符或预编译符填写失败的情况。12

故障表现：

编译时部分 sequence 处理函数无法识别导致报错，常见报错如下：13



	说明	文件	列	行	项目
4	error C2039: "clear"：不是 "DDS_StringSeq" 的成员	test_publication.cpp	1	58	test_publication
2	error C2039: "ensure_length"：不是 "DDS_StringSeq" 的成员	test_publication.cpp	1	56	test_publication
3	error C2039: "set_at"：不是 "DDS_StringSeq" 的成员	test_publication.cpp	1	57	test_publication
7	IntelliSense: class "DDS_StringSeq" 没有成员 "clear"	test_publication.cpp	50	58	test_publication
5	IntelliSense: class "DDS_StringSeq" 没有成员 "ensure_length"	test_publication.cpp	51	56	test_publication
6	IntelliSense: class "DDS_StringSeq" 没有成员 "set_at"	test_publication.cpp	47	57	test_publication
1	warning MSB8028: The intermediate directory (Debug\) contains files shared from another project (test_subscription.vcxproj). This can lead to incorrect clean and rebuild behavior.	Microsoft.CppBuild.targets	5	388	test_publication

错误列表 输出 查找结果 1 查找符号结果

14

图 33 预编译符缺失导致的编译失败报错图 15

排查方式： 16

此时需根据用户手册第十章表 10-2 检查在工程的项目->属性->C/C++->预处理器->预处理器定义中填写的预编译符是否正确。17

3.1.3 头文件与库不匹配 18

可能存在本地安装多个版本的 ZRDDS，在工程中头文件和库版本不一致的情况。19

故障表现：1

可能出现无法解析的外部命令或运行过程中异常崩溃等问题。2

排查方式：

重新检查所添加的头文件和链接库位置是否为同一版本。3

3.1.4 库与编译方式不匹配 4

可能存在编译方式与链接库不匹配情况，如链接动态库，使用静态编译或链接静态库，5使用 DLL 编译

故障表现：

可能出现未声明的标识符，无法解析的外部命令等问题。6

排查方式：

对照用户手册第十章表 10.1 检查当前编译方式所链接的库是否正确。7

3.1.5 动态库缺失 8

可能出现动态库路径未在链接路径（LD_LIBRARY_PATH）以及运行路径（Path）造成的9编译或运行失败；

故障表现：

在运行时报错无法启动，丢失 dll。10

排查方式：11

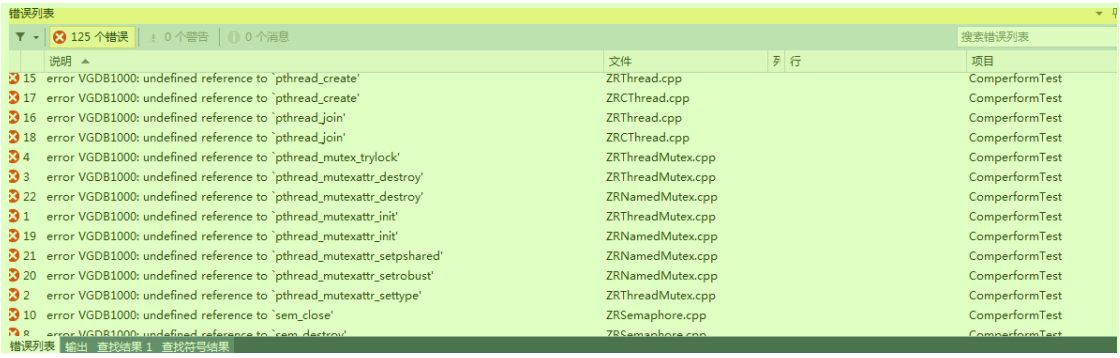
查看链接路径与运行路径下是否存在所需的动态库，若不存在，则将动态库拷贝到相应12位置，再次运行，查看编译情况。

3.1.6 VisualGDB 库缺失 13

在编译 GDB 工程时可能由于库缺失或库名填写错误导致编译失败。（注意：GDB 工程下14输入库文本的名字时需要去除 lib 部分。）

故障表现：

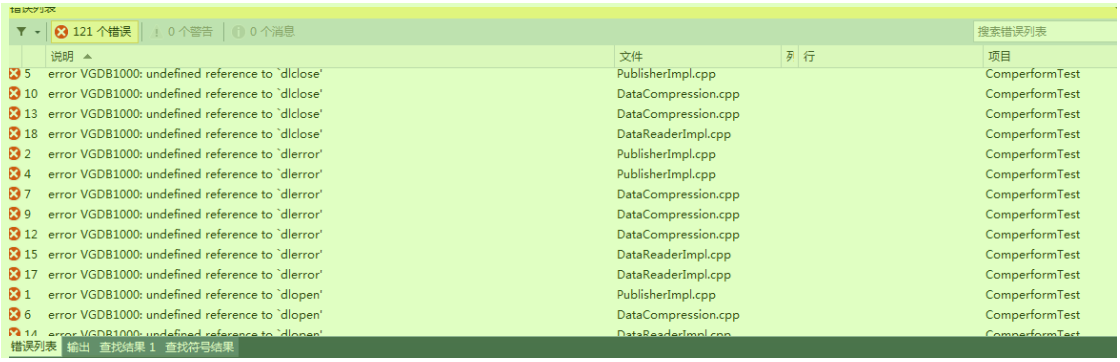
当缺失 pthread 库时通常会有以下报错：15



16

图 34 缺失 pthread 报错图 17

当缺失 dl 库时通常会有以下报错：18



	说明	文件	列	行	项目
5	error VGDB1000: undefined reference to `dlclose'	PublisherImpl.cpp			ComperformTest
10	error VGDB1000: undefined reference to `dlclose'	DataCompression.cpp			ComperformTest
13	error VGDB1000: undefined reference to `dlclose'	DataCompression.cpp			ComperformTest
18	error VGDB1000: undefined reference to `dlclose'	DataReaderImpl.cpp			ComperformTest
2	error VGDB1000: undefined reference to `dlerror'	PublisherImpl.cpp			ComperformTest
4	error VGDB1000: undefined reference to `dlerror'	PublisherImpl.cpp			ComperformTest
7	error VGDB1000: undefined reference to `dlerror'	DataCompression.cpp			ComperformTest
9	error VGDB1000: undefined reference to `dlerror'	DataCompression.cpp			ComperformTest
12	error VGDB1000: undefined reference to `dlerror'	DataCompression.cpp			ComperformTest
15	error VGDB1000: undefined reference to `dlerror'	DataReaderImpl.cpp			ComperformTest
17	error VGDB1000: undefined reference to `dlerror'	DataReaderImpl.cpp			ComperformTest
1	error VGDB1000: undefined reference to `dlopen'	PublisherImpl.cpp			ComperformTest
6	error VGDB1000: undefined reference to `dlopen'	DataCompression.cpp			ComperformTest

1

图 35 缺失 dl 库报错图 2

排查方式： 3

1) 查看是否添加 pthread 库。4
2) 查看是否添加 dl 库。5

3.1.7 丢失 Psapi.lib 6

Zrdds 版本> =2.3.3 7

在使用 VS2008、VS2010 编译和 WindowsXP 版本程序时，由于版本问题可能会缺少一个系统库 Psapi.lib；8

报错信息如下：9

```
"/LTCG" 规范)
版本\vs_2008_x32\ZRDDS\ZRDDS-2.3.3\examples\cpp\ide\Debug\DataReceiveByListener_Subscription_i86
: error LNK2001: 无法解析的外部符号 _EnumProcessModules@16
: error LNK2001: 无法解析的外部符号 _GetModuleFileNameExA@16
32\ZRDDS\ZRDDS-2.3.3\examples\cpp\ide\Debug\DataReceiveByListener_Subscription_i86Win32VS2008.ex
20220928\版本\vs_2008_x32\ZRDDS\ZRDDS-2.3.3\examples\cpp\ide\DataReceiveByListener_Subscription'
3 - 3 个错误，1 个警告
跳过 0 个 ==========
```

10

解决：在链接器里加 Psapi.lib 即可；11

3.1.8 VS2015 版本（update 3）和库不匹配 12

原因：用老版本的编译器链接新版本编译器编出来的库，报版本对不上；13

报错信息如下：

```
显示输出来源(S): 生成
1> 正在生成代码...
1> ZRDDSCpprd_VS2015.lib(ZRBuiltinTypeCPlusPlusSequence.obj) : 找到 MSIL .netmodule 或使用 /GL 编译的模块；正在使用 /LTCG 重新启动链接；将 /LTCG 添加到链接命令行以改
1>LINK : warning LNK4075: 忽略"/INCREMENTAL"(由于"/LTCG"规范)
1>main_pub.obj : warning LNK4075: 忽略"/EDITANDCONTINUE"(由于"/OPT:LBR"规范)
1>LINK : fatal error C1900: "P1"(第"20150812"版)和"P2"(第"20130802"版)之间 Il 不匹配
1>LINK : fatal error LNK1257: 代码生成失败
========== 生成: 成功 0 个，失败 1 个，最新 0 个，跳过 0 个 ==========
```

14

解决方案：15

1） 更新 VS2015 update 3 版本；16

3.2 初始化失败 1

3.2.1 报错 2

一般情况下，DDS 初始化失败会打印错误日志来提示失败的原因，以下为常见的初始化失败场景。 3

3.2.1.1 Licence 验证 4

```
zrdds try find zrddslicence.lic in ZRDDS_HOME(D:\WBY\tmp\ZRDDS\ZRDDS-2.2.7\) dir
zrdds can not find zrddslicence.lic file in current dir or ZRDDS_HOME(D:\WBY\tmp\ZRDDS\ZRDDS-2.2.7\) dir.
validate licence failed(-1).
DomainParticipantFactoryGetInstance failed.
```

5

图 36 licence 文件未找到 6

出现图 36 错误日志打印表示用户验证文件（zrddslicence.lic）没有找到。DDS 会在环境变量${ZRDDS_HOME}所表示目录和当前程序的运行目录下查找用户验证文件。因此，出现图 36 问题需要确认上述目录下是否有用户验证文件、用户验证文件是否被重命名等。 7

```
zrdds use zrddslicence.lic in current dir
hash not consist(0 -1), licence may be modified.
verify licence file(.\zrddslicence.lic) failed(-8).
validate licence failed(-8).
DomainParticipantFactoryGetInstance failed.
```

8

图 37 licence 文件被修改 9

出现图 37 错误日志打印表示 DDS 成功读取到了用户验证文件（zrddslicence.lic），但文件内容修改导致验证不成功。用户验证文件不允许修改，修改导致无法验证成功需要将文件修改还原，若无法还原则需向我司重新申请用户验证文件。 10

```
zrdds use zrddslicence.lic in current dir
ignore user verify with userName(UserName: wby)
Licence expired(Expire Date: 2021/11/25:15:04:40) now(Expire Date: 2022/03/16:11:49:19).
expire date(Expire Date: 2021/11/25:15:04:40) verify failed(-1).
verify licence file(.\zrddslicence.lic) failed(-10).
validate licence failed(-10).
DomainParticipantFactoryGetInstance failed.
```

11

图 38 licence 文件过期 12

出现图 38 错误日志打印表示当前系统时间不在用户验证文件的有效时间范围中，用户验证文件中的时间是按现实中的时间设置的，所以在使用时要确保系统时间和现实时间对应。 13

```
c@lc-virtual-machine:~/桌面/test_zero$ ./app4
rdds use zrddslicence.lic in current dir
nvalid licence content with(0x1896950 0x189695c 0x189697c (nil) (nil) (nil) 0x1
96acf), lic may be modified.
erify licence file(./zrddslicence.lic) failed(-6).
alidate licence failed(-6).
omainParticipantFactoryGetInstance failed.
omainParticipantFactoryGetInstance failed.
DSIF::Initialize failed.
```

1

图 39　licence 文件格式错误 2

如图 39 错误日志打印，当 lic 文件没有更改文本格式时，例如 windows 下的 lic 文件在 linux 下使用时同样也会报 lic 被修改的报错，并且返回值是-6; 在 notepad 里可以更改文本格式 3

3.2.1.2 网络环境 1

```
***************    Wed Mar 16 14:54:18 2022    ***************
LEVEL: 2
FILE: D:\WBY\ZRDDS_304\ZRDDS_C\src\ZRDDSCoreImplement\Locator_t.cpp
FUNC: LocatorListGetDefaultIpv4Addrs(LINE: 665)
loopback is not enabled.
*****************************************************************

***************    Wed Mar 16 14:54:18 2022    ***************
LEVEL: 2
FILE: D:\WBY\ZRDDS_304\ZRDDS_C\src\ZRDDSCoreImplement\Locator_t.cpp
FUNC: LocatorListFromStringSeq(LINE: 808)
LocatorListGetDefaultIpv4Addrs failed(-1).
*****************************************************************

***************    Wed Mar 16 14:54:18 2022    ***************
LEVEL: 2
FILE: D:\WBY\ZRDDS_304\ZRDDS_C\src\ZRDDSCoreImplement\QoSHelper.cpp
FUNC: DomainParticipantQosValid(LINE: 97)
process metatraffic_receive_addresses failed.
*****************************************************************

***************    Wed Mar 16 14:54:18 2022    ***************
LEVEL: 2
FILE: D:\WBY\ZRDDS_304\ZRDDS_C\src\ZRDDSCoreImplement\Locator_t.cpp
FUNC: LocatorListGetDefaultIpv4Addrs(LINE: 665)
loopback is not enabled.
*****************************************************************

***************    Wed Mar 16 14:54:18 2022    ***************
LEVEL: 2
FILE: D:\WBY\ZRDDS_304\ZRDDS_C\src\ZRDDSCoreImplement\Locator_t.cpp
FUNC: LocatorListFromStringSeq(LINE: 808)
LocatorListGetDefaultIpv4Addrs failed(-1).
*****************************************************************

***************    Wed Mar 16 14:54:18 2022    ***************
LEVEL: 2
FILE: D:\WBY\ZRDDS_304\ZRDDS_C\src\ZRDDSCoreImplement\QoSHelper.cpp
FUNC: DomainParticipantQosValid(LINE: 105)
process usertraffic_receive_addresses failed.
*****************************************************************
```

2

图 39 没有可用网卡 3

出现图 39 错误日志打印时，表示 DDS 没有找到可用的网卡，并且没有打开使用本地回环的选项。当使用场景为多台设备进行通信时，出现这个错误需要检查当前设备是否有可使用的网卡、网线连接是否正常。当使用场景为单台设备自发自收时，出现这个错误需要确定 qos 配置中是否禁止了本地回环的使用。 4

```
***************    Wed Mar 16 14:49:48 2022    ***************
LEVEL: 2
FILE: D:\WBY\ZRDDS_304\ZRDDS_C\src\ZRDDSCoreImplement\NetworkUtility.cpp
FUNC: ZRGetIPv4AddrNumbers(LINE: 243)
GetAdaptersInfo failed.
*****************************************************************

***************    Wed Mar 16 14:49:48 2022    ***************
LEVEL: 2
FILE: D:\WBY\ZRDDS_304\ZRDDS_C\src\ZRDDSCoreImplement\Locator_t.cpp
FUNC: LocatorListGetDefaultIpv4Addrs(LINE: 656)
ZRGetIPv4AddrNumbers failed(-1).
*****************************************************************

***************    Wed Mar 16 14:49:48 2022    ***************
LEVEL: 2
FILE: D:\WBY\ZRDDS_304\ZRDDS_C\src\ZRDDSCoreImplement\Locator_t.cpp
FUNC: LocatorListFromStringSeq(LINE: 808)
LocatorListGetDefaultIpv4Addrs failed(-1).
*****************************************************************

***************    Wed Mar 16 14:49:48 2022    ***************
LEVEL: 2
FILE: D:\WBY\ZRDDS_304\ZRDDS_C\src\ZRDDSCoreImplement\QoSHelper.cpp
FUNC: DomainParticipantQosValid(LINE: 97)
process metatraffic_receive_addresses failed.
*****************************************************************

***************    Wed Mar 16 14:49:48 2022    ***************
LEVEL: 2
FILE: D:\WBY\ZRDDS_304\ZRDDS_C\src\ZRDDSCoreImplement\NetworkUtility.cpp
FUNC: ZRGetIPv4AddrNumbers(LINE: 243)
GetAdaptersInfo failed.
*****************************************************************

***************    Wed Mar 16 14:49:48 2022    ***************
LEVEL: 2
FILE: D:\WBY\ZRDDS_304\ZRDDS_C\src\ZRDDSCoreImplement\Locator_t.cpp
FUNC: LocatorListGetDefaultIpv4Addrs(LINE: 656)
ZRGetIPv4AddrNumbers failed(-1).
*****************************************************************

***************    Wed Mar 16 14:49:48 2022    ***************
LEVEL: 2
FILE: D:\WBY\ZRDDS_304\ZRDDS_C\src\ZRDDSCoreImplement\Locator_t.cpp
FUNC: LocatorListFromStringSeq(LINE: 808)
LocatorListGetDefaultIpv4Addrs failed(-1).
*****************************************************************

***************    Wed Mar 16 14:49:48 2022    ***************
LEVEL: 2
FILE: D:\WBY\ZRDDS_304\ZRDDS_C\src\ZRDDSCoreImplement\QoSHelper.cpp
FUNC: DomainParticipantQosValid(LINE: 105)
process usertraffic_receive_addresses failed.
*****************************************************************
```

1

图 40 没有网卡 2

windows 出现图 40 错误日志打印时，表示 DDS 没有找到任何网卡。这种情况下需要检查是否将本机的网卡都禁用了。 3

3.2.2 崩溃 1

在使用 DDS 进行初始化过程中出现问题时，一般情况下只会进行错误日志打印，不会造成程序崩溃。为了分析确定具体的崩溃原因，以下为一些必要的检查措施： 2

确定 DDS 版本信息。默认情况下 DDS 的版本信息会在程序和日志文件中打印，是 DDS 相关的第一条日志打印。图 41 即为程序中的 DDS 版本信息打印，version<>中表示的是 DDS 版号，was compiled at 后面的时间表示当前使用的 DDS 库编译生成的时间。图 42 为 DDS 日志文件中的版本信息，DDS 日志文件以程序名作为文件名，以 ddslog 作为文件后缀，如 test.exe.ddslog。DDS 日志文件会保存在程序运行的目录下（与程序所在目录进行区分）。 3

```
****************    Wed Mar 16 15:38:14 2022    ****************
Current ZRDDS version(2.2.7) was compiled at Mar 16 2022 13:56:56
*************************************************************
```

4

图 41 程序中的 DDS 版本信息打印 5

```
****************    Wed Mar 16 15:38:14 2022    ****************
Current ZRDDS version(2.2.7) was compiled at Mar 16 2022 13:56:56
*************************************************************
```

6

图 42 日志文件中的 DDS 版本信息打印 7

检查工程配置。参照 ZRDDS 安装配置手册，检查相应的工程配置过程是否有误或遗漏。 8

检查头文件和库是否匹配。通过查看程序所使用的库和头文件 ZROSDefine.h（ZRDDSCoreInterface 目录下）的修改时间，可以简单判断两者是否匹配。

检查是否判断返回值。在 3.2.1 中，DDS 初始化失败打印错误日志之后，函数调用会返回空指针不会是成功之后的 DDS 对象。如果对返回值没有进行空指针判断而直接进行后续操作，就会造成崩溃问题。 9

3.2.3 其他 10

3.2.3.1 linux 验证卡死 11

linux 上在启动 DDS 时，可能会遇到程序卡在 licence 验证，无法正常运行下去。这种情况可能是由于之前一次程序启动时，在进行 licence 验证时，程序提前退出导致有个信号量没有销毁，之后再启动程序会获取到之前的信号量，卡在等待信号量释放。 12

验证和解决方法如图 43 所示，到 /dev/shm 目录下，查看是否有 sem.ZRDDSUserValidatorSemaphore 这个文件，将这个文件删除即可。 13

```
root@ht706-os:~# cd /dev/shm
root@ht706-os:/dev/shm# ls
pulse-shm-1015282209  pulse-shm-2038469434  pulse-shm-3115712205  pulse-shm-4251534699
pulse-shm-1636677703  pulse-shm-234970151   pulse-shm-3354595245  pulse-shm-867520997
pulse-shm-1757609557  pulse-shm-2470270404  pulse-shm-3377139384  sem.ZRDDSUserValidatorSemaphore
pulse-shm-1886373333  pulse-shm-2971836217  pulse-shm-3921990753
pulse-shm-1887924241  pulse-shm-2986750821  pulse-shm-3962749238
root@ht706-os:/dev/shm# rm sem.ZRDDSUserValidatorSemaphore
root@ht706-os:/dev/shm# ls
pulse-shm-1015282209  pulse-shm-2038469434  pulse-shm-3115712205  pulse-shm-4251534699
pulse-shm-1636677703  pulse-shm-234970151   pulse-shm-3354595245  pulse-shm-867520997
pulse-shm-1757609557  pulse-shm-2470270404  pulse-shm-3377139384
pulse-shm-1886373333  pulse-shm-2971836217  pulse-shm-3921990753
pulse-shm-1887924241  pulse-shm-2986750821  pulse-shm-3962749238
root@ht706-os:/dev/shm#
```

1

图 43 销毁信号量 2

3.3 收不到数据 3

3.3.1 网络环境检测 4

（1） 使用 ping 命令查看相关节点网络是否联通（具体操作参见 2.1）; 5

（2） 使用 iperf/sockperf 进行组播收发测试（具体操作参见 2.1）: 6

a) 多网卡情况下的组播测试; 7

b) 如果不通需检查组播路由、防火墙设置; 8

（3） 操作系统是否安装了安全、杀毒软件，如 360 安全卫士/360 杀毒，江民等。如有，请彻底退出并卸载后再次尝试。或将 DDS 应用程序设置为软件的白名单; 9

（4） 网络内是否有相同 IP 的节点，常见如：本机有多个网卡或多个 IP，分别参与了不同子网，而其它网内有相同的 IP 节点。 10

如目前大部分银河麒麟机器上预装了 docker，这导致每个机器上都有 docker0 网卡，且不同机器上的 docker0 网卡大概率被预设为默认一致的地址（默认为 172.12.0.1）。DDS 可能会误用该网卡的地址导致数据发送至这个 docker 网卡的地址中，最终导致无法通信。这种情况下，建议通过以下方式进行处理：

a) 修改 IP 地址; 11

b) 通过可配置性屏蔽指定网卡; 12

i. 通过配置文件修改 qos 13

```
<participantfactory_qos>
<property>
<value>
<element>
<name>sysctl.global.network.disabled_interface</name>
<value>docker0</value>
<propagate>false</propagate>
</element>
</value>
</property>
</participantfactory_qos>
```

14

ii. 通过代码修改 qos

```
stringethernetName="docker0";

DomainParticipantFactoryQosdpfQos;
TheParticipantFactory->get_qos(dpfQos);
DDS_PropertyQos_AppendProperty(&dpfQos.property,"sysctl.global.network.
TheParticipantFactory->set_qos(dpfQos);
```

（5） 运行安装包下的测试例子验证是否能够运行成功；2

3.3.2 ZRDDS 配置检测 3

（1） 是否相同域 4

5

a) 检查代码

```
DDS::DomainParticipant *participant = factory->create_participant(
        domainId,
        dpQos,
        NULL,
        DDS::STATUS_MASK_NONE);
```

图 44 创建域参与者 6

b) 监控器查看域号信息 7

8

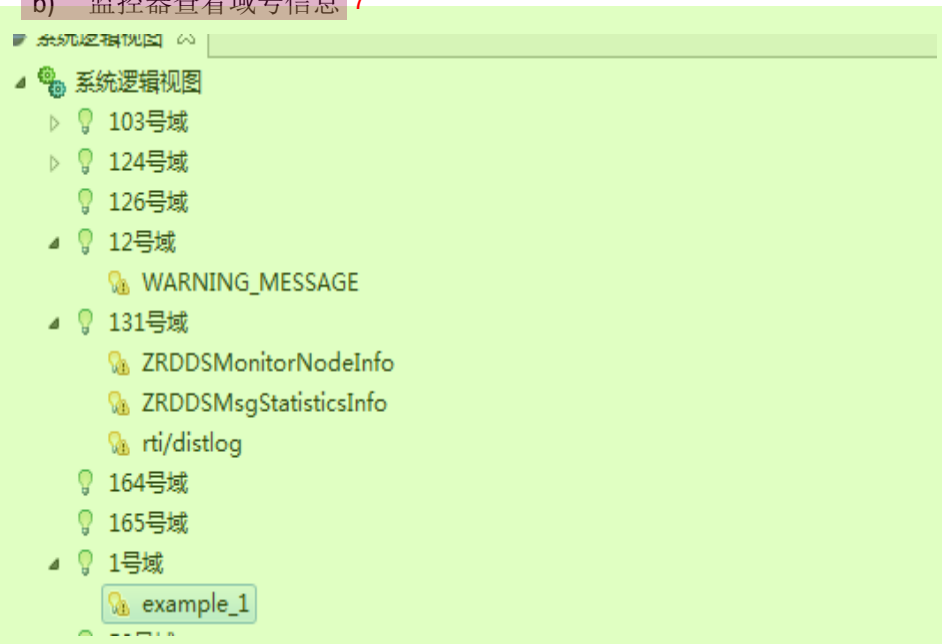


图 45 监控器域视图 9

（2） 主题名、类型名是否匹配 10

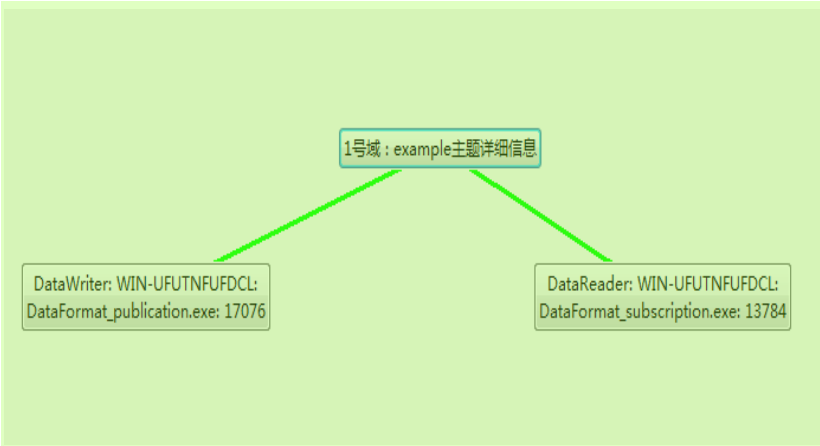
a) 检查代码 11

```
DDS::Topic *topic = participant->create_topic(
        "example",
        typeName,
        DDS::TOPIC_QOS_DEFAULT,
        NULL,
        DDS::STATUS_MASK_NONE);
```

1

图 46 创建主题 2

b) 监控器查看主题信息（截图）；3



4

图 47 主题信息 5

（3） QoS 是否兼容 6

a) 检查代码 7

```
DDS::DataWriterQos writerQos;
publisher->get_default_datawriter_qos(writerQos);
//TODO 在这里修改数据写者Qos
writerQos.history.depth = 5;
DDS::DataWriter *writer = publisher->create_datawriter(
        topic,
        writerQos,
        NULL,
        DDS::STATUS_MASK_NONE);
```

8

图 48 创建数据写者 9

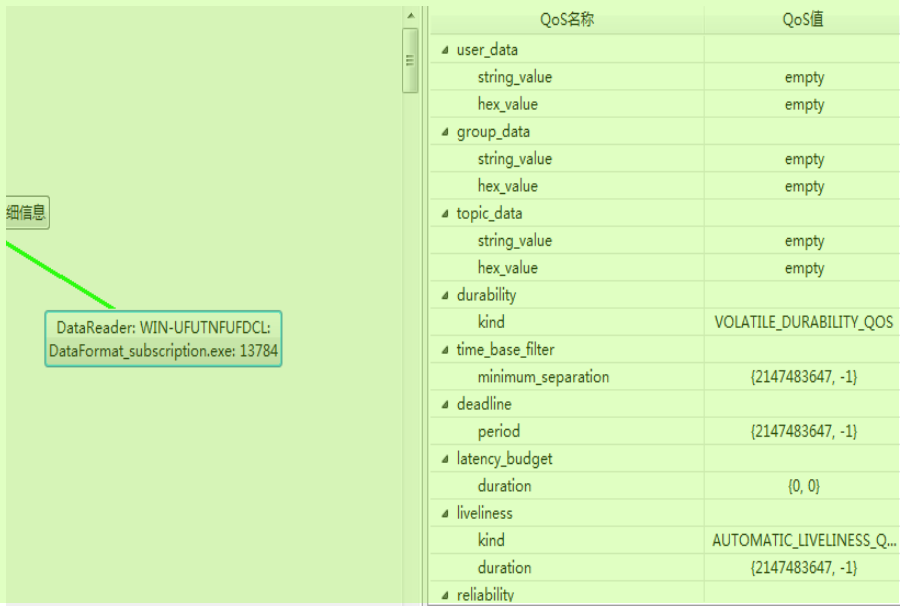
b) 查看是否有 QoS 不匹配的日志信息；10

```
LEVEL: 16
FILE: ..\..\..\src\ZRDDSCoreImplement\DataWriterImpl.cpp
FUNC: DataWriterQosCompatibleCheck(LINE: 8139)
Match Reader(2) (6397a8c0)(04e90000)(00000000)(0e000007) failed: Reliability inc
ompatible in DataWriter(1) (6397a8c0)(dc9c0000)(00000000)(0e000002) of Topic Pos
it.
*************************************************************
```

11

图 49 Qos 不匹配打印 12

c) 监控器查看实体信息（截图）；1



2

图 50 实体 Qos 信息视图 3

（4） 配置的地址不正确 4

a) 配了两个不同的网段 5

```
DDS::DomainParticipantQos dpQos;
factory->get_default_participant_qos(dpQos);
const char* user_addr = "tcpv4_raw://192.168.156.12//8800";
dpQos.usertraffic_receive_addresses.addresses.ensure_length(1, 1);
dpQos.usertraffic_receive_addresses.addresses.set_at(0, user_addr);
// TODO 在这里修改DomainParticipantQos
```
6

图 51 配置域参与者使用的地址 7

b) 自动地址排序是否正确（查看日志）8

日志号（49），参见 2.2.3 9

3.3.3 高级诊断 10

如果环境、ZRDDS 配置检测均正确，且发送成功：11

1、 确认网络中确实有数据发送至本节点。抓包确认发送方节点是否有向本节点发送数据，且指向端口正确，且该端口被 DR 所在进程正确地监听。如果没有，需要考虑发送是否成功或者节点之间的网络问题； 12

抓包分析过滤条件：rtps && ip.addr==192.168.1.1 && ip.addr==192.168.1.2

2、 可靠传输的情况下，可以查看发送方节点向本节点发送的数据中，是否有 DATA 或 DATA_FRAG 的消息。如果有，可以通过 writerSeqNumber 查看这些 DATA/DATA_FRAG 是否一直是同一包； 13

```
submessageId: DATA (0x15)
  Flags: 0x05, Data present, Endianness bit
  octetsToNextHeader: 64
  0000 0000 0000 0000 = Extra flags: 0x0000
  Octets to inline QoS: 16
  readerEntityId: 0x0b000004 (Application-defined reader (no key): 0x0b0000)
  writerEntityId: 0x0d000003 (Application-defined writer (no key): 0x0d0000)
  writerSeqNumber: 3
  serializedData
```

图 52 DATA 子消息

如果一直是同一包，则可以考虑是否用户未取数据。或是 DataReader 中有关 ResourceLimit 的处理存在异常，这种情况下，需要使用日志进一步进行分析。

3、 可靠传输的情况下，如果抓包中未看到 DATA 或 DATA_FRAG 类型的子消息，只能看到 HEARTBEAT、ACK_NACK 或 NACK_FRAG。则可以：

a) 查看通过 firstAvailableSeqNum 及 lastSeqNum 来确认 HEARTBEAT 的数据范围。HEARTBEAT 是数据写者表明自身保存的数据序号范围的子消息。正常情况下，如果开启了内存持久化，则持有数据量不超过 resourceLimitQos 中设置的 max_samples_per_instance 的数值；如果未开启内存持久化，则不超过 historyQos 中 depth 的数值。如果 firstAvailableSeqNum>lastSeqNum，说明数据写者中已经没有数据，请确认数据写者调用 write 时的返回值及网络环境状况；

```
submessageId: HEARTBEAT (0x07)
  Flags: 0x01, Endianness bit
  octetsToNextHeader: 28
  readerEntityId: 0x0b000004 (Application-defined reader (no key): 0x0b0000)
  writerEntityId: 0x0d000003 (Application-defined writer (no key): 0x0d0000)
  firstAvailableSeqNumber: 3
  lastSeqNumber: 3
  count: 5
```

说明该DW的内存中的数据范围为[3,3]

图 53 HEARBEAT 子消息

b) 如果确认 HERATBEAT 没有异常，则可以确认数据读者的相关状况。

ACKNACK 分为两种类型，一种是表明“ACK”，告知数据写者已收到，可以继续往下发。还有一种表明“NACK”，即告知数据写者自身确实的数据，请求重传。

如下图所示为“ACK”。bitmapBase 为 4，numBits 为 0，表明该数据读者已经收到了 SN=3 的数据，可以开始接收 SN=4 的数据了。

```
submessageId: ACKNACK (0x06)
  Flags: 0x03, Final flag, Endianness bit
  octetsToNextHeader: 24
  readerEntityId: 0x0b000004 (Application-defined reader (no key): 0x0b0000)
  writerEntityId: 0x0d000003 (Application-defined writer (no key): 0x0d0000)
  readerSNState
    bitmapBase: 4
    numBits: 0
    [Acknack Analysis: Expecting sample 4]
  Count: 6
```

表明该读者已经收到SN=3的数据
可以开始接收SN=4的数据了

图 54 ACKNACK 子消息 “ACK”

如下图所示，为“NACK”。bitmapBase 为 103454，numBits 为 1，表明从

103454 起始的第 1 包丢失，请求重传。

```
◢ submessageId: ACKNACK (0x06)
  ▷ Flags: 0x01, Endianness bit
    octetsToNextHeader: 28
  ▷ readerEntityId: 0x1c000004 (Application-defined reader (no key): 0x1c0000)
  ▷ writerEntityId: 0x15000003 (Application-defined writer (no key): 0x150000)
  ◢ readerSNState
      bitmapBase: 103454
      numBits: 1
      [Acknack Analysis: Lost samples 103454 in range [103454,103454]]
      bitmap: 1
```

表明该数据读者未收到SN=103454的数据
要求数据写者重传

图 55ACKNACK 子消息“NACK”

注意，数据读者的“NACK”所请求的 SN 理应在数据写者所发的 HEARTBEAT 范围之内，“ACK”则应比数据写者的 HEARTBEAT 的 lastSeqNum 大 1，否则说明数据读者处理 HEARTBEAT 的过程存在异常。这种情况下，需要使用日志进一步进行分析。如果 ACKNACK 没有异常，且一直处于“ACK”的状态，则需要确认数据写者的数据是否发送成功？或考虑数据写者的数据发送流程异常，此时，需要进一步使用日志进行确认。

c) 如果 ACKNACK 中，存在“NACK”类型的消息。则需要检查数据读者向数据写者发回 NACK 后，数据写者是否有向数据读者重发该 NACK 所指向的 SN 的数据（DATA/DATA_FRAG），或是否有向数据读者发送 GAP。如果以上都无，而是周期性（默认为 3 秒）继续发送与之前一模一样的 HEARTBEAT，则需要考虑数据写者处理 HEARTBEAT 的逻辑异常，需要进一步使用日志进行分析。

4、 使用日志进行分析。详见 2.2.3 常用调试日志号。

3.4 通信异常

3.4.1 序列化/反序列化失败

```
***************    Wed Mar 16 16:46:59 2022    ***************
LEVEL: 2
FILE: C:\ZRDDSPublisher\build\project\zrdds\src\ZRDDSCoreImplement\MessageBlock.cpp
FUNC: MessageBlockWriteData(LINE: 99)
assert MessageBlockTotalSpace(256) < length(300) failed.
MessageBlock m_base(02ce2be0), m_wrPtr(8), m_cont(00000000)
00 01 00 00 2c 01 00 00
*****************************************************************
serialize sample->value failed.
***************    Wed Mar 16 16:46:59 2022    ***************
LEVEL: 2
FILE: C:\ZRDDSPublisher\build\project\zrdds\src\ZRDDSCoreImplement\DataWriterImpl.cpp
FUNC: DataWriterSerializeSample(LINE: 5672)
Write failed: failed to serialize data(2):
frags: 1, sample max length: 259,
sample length: 304
in DataWriter (059aa8c0)(d0170000)(00000000)(0a000003) of topic example
*****************************************************************
write data failed.
```

图 56 序列化失败

图 56 为 DDS 发送方发送时序列化数据失败，从而导致发送失败。图 56 中的序列化失败的原因是给 sequence 设置的大小超过了最大可设置大小 256B。通过 idl 生成自定义数据

类型时，默认情况下（zrddsgen 没有加-u 参数）sequence 最大为 256B。zrddsgen 生成时加上-u 参数，则 sequence 大小可以设置大于 256B。

```
***************    Thu Mar 17 17:22:18 2022    ***************
LEVEL: 2
FILE: C:\ZRDDSPublisher\build\project\zrdds\src\ZRDDSCoreImplement\DataReaderImpl.cpp
FUNC: DataReaderImplReadOrTakeNoKey(LINE: 1612)
reader(<059aa8c0><b8320000><00000000><0a000004> example test> deserialize sample failed(-3).
*****************************************************************
```

图 57 反序列化失败

图 57 中的错误日志表示数据接收时数据反序列化失败，出现这个问题的原因可能是收发两边数据类型同名但内部结构不同。根据最后返回的错误码可以在 idl 生成的文件中对应反序列化函数内查看具体的失败原因。

3.4.2 丢包严重

使用 DDS 出现丢包严重问题的原因大致可以分为两部分：

- 处理耗时。在接收程序中，如果在接收回调中进行长时间处理，就可能导致数据接收不及时而丢失。在接收回调中尽量只做一些数据拷贝存储的操作，将耗时的处理在其他地方其他线程进行。
- 网络问题。出现丢包或者使用 DDS 之前，最好测试一下网络本身的丢包情况，如果网络本身丢包就很严重，DDS 也就无法确保能够完全可靠不丢包。同时网络丢包除了直接影响数据通信，还可能影响程序之间 DDS 维持匹配的机制，出现反复断开匹配也会产生丢包的现象。

3.4.3 数据中断

通信过程中出现数据中断问题，在没有错误日志打印的情况下，可能的原因为 dp 超时导致收发匹配断开。

判断是否存在 dp 超时可以通过使用交互式日志的方法，参考交互式日志的使用方法，打开 dp 超时下线的日志号，观察数据中断时是否有相应日志打印。

dp 超时下线是因为在一定时间内没有收到 dp 的发现数据。由于发现数据的收发是不可靠的，发现数据就可能因为网络问题频繁丢包导致判断下线。另一方面如果 dp 的发现数据发送频率过低，而超时时间又相对较小，则轻微的网络问题可能导致发现数据少量丢包就判断超时，更容易判断下线。

3.4.4 发送失败

DDS 发送失败的原因需要根据实际错误日志来分析，如 3.4.1 中序列化问题会导致发送失败，以下为常见的发送失败情况。

使用可靠模式进行 udp 通信时，如果接收方在接收回调的处理耗时过长，可能导致发送方发送队列中的数据无法及时确定已接收，队列数据来不及清理，从而使新的数据无法加入队列发送，数据发送失败，出现图 58 中警告日志。

```
***************    Thu Mar 17 14:24:07 2022    ***************
LEVEL: 16
FILE: /home/ht706/wby/ZRDDS_C/src/ZRDDSCoreImplement/DataWriterImpl.cpp
FUNC: obtainSample(LINE: 1655)
Waiting sample (sn:51) space failed: 10 in instance null in DataWriter (66aca8c0)(971d0000)(00000000)(0b000003) of topic Test1
*********************************************************

***************    Thu Mar 17 14:24:24 2022    ***************
LEVEL: 16
FILE: /home/ht706/wby/ZRDDS_C/src/ZRDDSCoreImplement/DataWriterImpl.cpp
FUNC: obtainSample(LINE: 1655)
Waiting sample (sn:68) space failed: 10 in instance null in DataWriter (66aca8c0)(971d0000)(00000000)(0b000003) of topic Test1
*********************************************************
```

图 58 发送队列已满

3.4.5 网络中断

使用 TCP 协议通信时，由于网络中断导致通信异常会有相应的日志信息。当网线连接异常或其他原因导致网络中断时，数据发送超时，会尝试 TCP 重新连接，直到重新连接成功，如图 59、图 60、图 61、图 62。

```
****************    Thu Mar 17 14:46:29 2022    ****************
LEVEL: 2
FILE: D:\WBY\ZRDDS_304\ZRDDS_C\src\ZRDDSCoreImplement\DataWriterImpl.cpp
FUNC: DataWriterProcSendFailed(LINE: 13349)
(Test1) Send message failed:18, reconnecting.
*****************************************************************
```

图 59 发送失败（超时）

```
****************    Thu Mar 17 14:55:02 2022    ****************
LEVEL: 8
FILE: D:\WBY\ZRDDS_304\ZRDDS_C\src\ZRDDSCoreImplement\DataWriterImpl.cpp
FUNC: DataWriterTcpReconnEvent(LINE: 13479)
TCP locator reconnecting, waiting for it, remote addr 192.168.172.102:35161.
*************************************************************
```

图 60 尝试重连

```
****************    Thu Mar 17 14:55:03 2022    ****************
LEVEL: 2
FILE: D:\WBY\ZRDDS_304\ZRDDS_C\src\ZRDDSCoreImplement\DataWriterImpl.cpp
FUNC: DataWriterTcpReconnEvent(LINE: 13535)
TCP locator reconnect failed 0, retrying, remote addr 192.168.172.102:35161.
*************************************************************
```

图 61 重连失败

```
****************    Thu Mar 17 14:55:07 2022    ****************
LEVEL: 8
FILE: D:\WBY\ZRDDS_304\ZRDDS_C\src\ZRDDSCoreImplement\DataWriterImpl.cpp
FUNC: DataWriterTcpReconnEvent(LINE: 13518)
TCP locator reconnect successfully, remote addr 192.168.172.102:35161.
*************************************************************
```

图 62 重连成功

在通信过程中如果将网卡禁用，会导致 dp 的组播发现数据发送失败，出现图 63 中的错误日志。

```
***************     Thu Mar 17 15:16:39 2022     ***************
LEVEL: 2
FILE: D:\WBY\ZRDDS_304\ZRDDS_C\src\ZRDDSCoreImplement\NetLocatorInfo.cpp
FUNC: NetLocatorSendMsg(LINE: 1787)
local((059aa8c0)(cc0f0000)(00000000)(000001c1)) send to multicast(type(5) port(2
0105) addr(0100ffef 00000000 00000000 00000000) cpuId(0)) on (0) locator failed(
37)
*****************************************************************

***************     Thu Mar 17 15:16:39 2022     ***************
LEVEL: 2
FILE: D:\WBY\ZRDDS_304\ZRDDS_C\src\ZRDDSCoreImplement\BuiltinParticipantWriter.c
pp
FUNC: BuiltinParticipantWriterSendParticipant(LINE: 636)
local((059aa8c0)(cc0f0000)(00000000)) send data(p) to pdpAddr(length(1) type(5)
port(4e89) addr(100ffef 00 00 00) cpuId(0) ) failed(-5).
*****************************************************************

***************     Thu Mar 17 15:16:39 2022     ***************
LEVEL: 2
FILE: D:\WBY\ZRDDS_304\ZRDDS_C\src\ZRDDSCoreImplement\BuiltinParticipantWriter.c
pp
FUNC: BuiltinParticipantWriterResendTimerCallback(LINE: 768)
local((059aa8c0)(cc0f0000)(00000000)) periodically send data(p) failed(-1).
*****************************************************************
```

图 63 组播发送失败

3.5 异常崩溃

检查异常崩溃通常分为以下情况：

1.读取未赋值的变量；

（1）一个变量未初始化、未赋值，就读取它的值。

2.函数栈溢出

（1）定义了一个体积太大的局部变量

（2）函数嵌套调用，层次过深（如无穷递归）

3.数组访问越界

（1）访问数据元素时，下标越界

4.指针的目标对象不可用

（1）空指针

（2）野指针

- 指针未赋值
- free/delete 释放了的对象
- 不恰当的指针强制转换
-

Windows 下（以 VS 为例）：

第一步：根据报错值，初步判断是哪种类型的崩溃

“The variable is being used without being intialized.”代表读取未赋值的变量。

“Stack overflow 代表函数栈溢出 1

“Stack around the variable was corrupted”代表数组访问越界 2

“未处理的异常：0xC0000005:读取位置 0x00000000 时发生访问冲突”代表指针的目标对象不可用 3

第二步：在 debug 模式下进行调试，当程序运行崩溃时，如图 64 所示在界面中点击“中断”按钮。 4

5

图 64 程序崩溃图 6

第三步：打开“调用堆栈”窗口，这个窗口可以直接观察到发生错误的时候、函数栈的各层函数的信息。（如果没有显示这个窗口，可从菜单里“调试|窗口|调用堆栈”里打开）。 7

“调用堆栈”窗口里可以观察到： 8

- 函数的调用层次 9
- 每一次函数里的局部变量（含参变量）的值 10
- 全部变量的值 11

第四步： 12

1.若是函数的栈越界，需要一步一步来调试，并配合系统自带的“任务管理器”来确定是哪些步骤导致了内存快速上涨，并分析上涨原因。 13

2.若是野指针，则需要一步一步进行调试，来查看该指针在哪里被改变了。 14

Linux: 15

若现场有程序源码，可以借用 Visual GDB 进行调试（具体使用详见“调试器”使用章节）；若没有则需要使用 core dump + gdb 的方法来查找原因。 16

第一步：保存 core 文件，两种保存方式 17

1.在程序启动前，输入“ulimit -c unlimited”代码，此命令只会对当前终端有用。 18

2.修改/etc/security/limits.conf 文件，在文件中增加一行 19

```
# /etc/security/limits.conf
#
#Each line describes a limit for a user in the from
```
20

```
#
#<domain><type><item><value>
*    soft    core    unlimited
```

1

第二步：运行程序。当崩溃后，找到 core 文件（默认情况下，core 文件在执行文件所在目录下，名字为 core），查看 core 文件的大小，若文件过大，需要考虑是不是函数栈溢出；2

第三步：使用“gdb program core”加载 core 文件，其中 program 为可执行程序名，core 为生成的 core 文件名；3

第四步：加载 core 文件之后，系统一般会打印如所示的信息，通过查看信号量，来初步判断问题原因。4

```
GNU gdb (Ubuntu 9.1-0kylin1) 9.1
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "aarch64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./test...
[New LWP 1183328]
Core was generated by `./test'.
Program terminated with signal SIGABRT, Aborted.
```

5

图 65 GDB 加载 core 文件 6

注：程序崩溃，常见的信号量为 sinal SIGABRT(6)和 sinal SIGSEGV(11)。SIGABRT 一般是多次调用 free、fclose 等函数造成的；SIGSEGV 代表是非法内存访问，有可能是函数的栈溢出，使用空指针等造成，需要进一步分析。7

第五步：使用 bt 命令查看函数调用栈信息，常用命令如下：8

“bt”是”backtrace”的缩写，使用“bt”或者“backtrace”都可以 9
bt<n>：n 是一个正整数，表示只打印栈顶上 n 层的信息；
bt<-n>：-n 表示一个负整数，表示只打印栈底下 n 层的信息。

第六步：使用 f 命令查看某栈层的信息，常用命令如下：10

f <n>：n 是一个从 0 开始的整数，表示栈中层的编号

第七步：使用 info 命令查看详细信息，常用命令如下：11

info f：打印详细的当前栈层的信息 12
info args：打印出当前函数的参数名及其值
info locals：打印出当前函数中所有局部变量及其值

第八步：使用 GDB 进行调试，常用命令如下：13

(gdb) help：查看命令帮助，具体命令查询在 gdb 中输入 help+命令，简写 h 14
(gdb) run：重新开始运行文件(run-text：加载文本文件；run-bin：加载二进制文件)，简写 r
(gdb) start：单步执行，运行程序，停在第一执行语句

(gdb) list：查看源代码（list-n：从第 n 行开始查看代码；list+函数名：查看具体函数），简写 l 1

(gdb) set：设置变量的值

(gdb) next：单步调试（逐过程，函数直接执行），简写 n

(gdb) step：单步调试（逐语句，跳入自定义函数内部执行），简写 s

(gdb) backtrace：查看函数的调用的栈帧和层级关系，简写 b

(gdb) frame：切换函数的栈帧，简写 f

(gdb) info：查看函数内部局部变量的数值，简写 i

(gdb) finish：结束当前函数，返回到函数调用点

(gdb) continue：继续执行，简写 c

(gdb) print：打印值及地址，简写 p

(gdb) quit：退出 gdb，简写 q

(gdb) break+num：在第 num 行设置断点，简写 b

(gdb) info breakpoints：查看当前设置的所有断点

(gdb) delete breakpoints num：删除第 num 个断点，简写 d

(gdb) display：追踪查看具体变量的值

(gdb) undisplay：取消追踪观察变量

(gdb) watch：被设置观察点的变量发生修改时，打印显示

(gdb) i watch：显示观察点

(gdb) enable breakpoints：启用断点

(gdb) disable breakpoints：禁用断点

(gdb) x：查看内存

(gdb) run argv[1] argv[2]：调试时命令行传参

3.6 错误日志 2

3.6.1 日志索引 3

4

日志信息	章节号
local(%s) can not find available tcp/ip locators for remote(%s) in (%d %u) netRet(%d)	3.6.2
serialize * failed	3.6.3
domainId(%u) participant(%s) deserialize RTPSMsgHeader from srcAddr(%s) failed(%d).	3.6.4
Failed to set priority option for locator of Publisher %s, error %d.	3.6.5
Parse XML failed: %s.	3.6.6
Reliability has been disabled with ZeroCopyBytes writer %s of Topic %s.	3.6.7
local(%s) no availiable port under domain(%d), release port or specific port and retry.	3.6.8

3.6.2 无法找到对端可用地址 1

3.6.2.1 日志信息 2

```
********************  2022-08-16 10:17:28:964  ********************
LOGINFO: LEVEL(WARNING)
FUNC: ZRSpdp.cpp:ZRSpdpSortLocatorList(LINE: 1282)
local((151aa8c0)(74b80000)(00000000)) can not find available tcp/ip locators for
 remote((181aa9c0)(ec040000)(00000000)(000001c1)) in (1 0) netRet(0)
*****************************************************************
```

4

local(%s) can not find available tcp/ip locators for remote(%s) in (%d %u) netRet(%d) 3

本地运行的 DDS 程序无法找到与远程 DDS 程序有效 IP 进行通信，远程 IP 的信息参见 remote 后括号内的 16 进制数，图中转化为 IP 地址为 192.169.26.24。 5

3.6.2.2 原因 6

DDS 通过组播发现一个远端节点，远端 DDS 所在节点存在多个 IP 的情况下，DDS 会通过尝试 TCP 连接选择可用的 IP 地址进行通信。本地节点与远端节点间不存在能够通信的 IP 地址时（地址均不在同一个网段）会打印本条日志。 7

此外，与远程的域参与者进行地址选择的过程的最长时间为可配置性 "sysctl.global.net.auto_sort_timeout"（默认 1s）,当网络繁忙等原因导致在超时时间内未完成地址选取时也会报错。 8

3.6.2.3 可能导致的问题 9

（1） 未能正确选取远程 DDS 域参与者地址可能会导致无法传输数据。尝试建立连接的过程中； 10

（2） 如果连接失败通信线程最多会阻塞 1s，当存在多个不能连接的节点时，可能会影响本节点数据的收发。 11

3.6.2.4 解决方法 12

（1） 配置通信双边节点 IP 到同一个网段。13

（2） 通过设置域参与者的 meta_traffic_address 以及 user_traffic_address 为指定网段，此时 DDS 认为该节点上只有 1 个 IP 时，不会尝试使用 TCP 建立连接。 14

（3） 当网络较差的情况下，通过可配置性适当延长地址选择时间。15

（4） 域隔离，对不同网段的 DDS 程序使用不同的域号，域参与者之间就不会尝试建立连接。 16

3.6.3 序列化失败 1

3.6.3.1 日志信息 2

输出”serialize * failed“，序列化 sequence 类型数据失败。3

```
********************  2022-04-19 15:28:01:643  ********************
LOGINFO: LEVEL(ERROR)
FUNC: MessageBlock.cpp:MessageBlockWriteData(LINE: 99)
assert MessageBlockTotalSpace(256) < length(1024) failed.
MessageBlock m_base(02bc6020), m_wrPtr(16), m_cont(00000000)
00 01 00 00 00 00 00 00 00 00 00 00 04 00 00
**********************************************************************
serialize sample->payload failed.********************  2022-04-19 15:28:01:643  ***************
LOGINFO: LEVEL(ERROR)
FUNC: DataWriterImpl.cpp:DataWriterSerializeSample(LINE: 5723)
Write failed: failed to serialize data(2):
frags: 1, sample max length: 267,
sample length: 1036
in DataWriter (0d03a8c0)(e4b50000)(00000000)(0a000003) of topic perf_test_tpc_round_1
**********************************************************************
[ERR] src/rtcl.c:1010:rtcl_write(): ZRDDS failed to write, error code: 3
rtcl_write error [loop: 0]
^C
phx@kylin:rtcl_without_shm$
```

4

图 66 序列化 sequence 数据失败 5

3.6.3.2 原因 6

用户发送 sequence 数据长度大于 idl 设置的最大长度，如果 idl 中没有设置最大长度则编译器使用默认值 256. 7

检查，确认：struct 名.cpp 文件打开找到 StringSeqStructGetSerializedSampleMaxSize()，return 256 之类的； 8

加完-u 之后：9

```
}

DDS_ULong StringSeqStructGetSerializedSampleMaxSize()
{
    return MAX_UINT32_VALUE;
}

DDS_ULong StringSeqStructGetSerializedKeyMaxSize()
{
    return MAX_UINT32_VALUE;
}
```

10

3.6.3.3 解决方法 1

以下两种方法：2

（1） 设置较大长度限制，发送数据时不要超过长度限制。3

（2） 编译器选项设置不限制 sequence 长度，使用编译器时加入“-u”例：4

```
zrddsgen.exe -i a.idl -d . -l C++ -u
```

3.6.4 发现不一致版本 5

3.6.4.1 日志信息 6

domainId(%u) participant(%s) deserialize RTPSMsgHeader from srcAddr(%s) failed(%d). 7

收到从源 IP 地址的 RPTS 报文，报文头解析失败。8

3.6.4.2 原因 9

网络中其他程序使用了非 ZRDDS 的其他 DDS 产品，且该 DDS 产品不符合 OMG 的 RTPS 协议规范。10

该现象不会影响 ZRDDS 的正常通信。11

3.6.4.3 解决方法 12

以下两种方法：13

（1） 关闭非 ZRDDS 的程序或者在网络上进行物理隔离；14

（2） 使用符合 RTPS 协议规范的 DDS 产品。15

3.6.5 设置优先级失败 16

3.6.5.1 日志信息 17

Failed to set priority option for locator of Publisher %s, error %d. 18

Linux 操作系统上设置优先级选项失败。19

3.6.5.2 原因 20

该问题只会在 Linux 系统出现，设置 socket 通信优先级时需要 root 权限，普通用户启动程序时就会报错。21

3.6.5.3 解决方案 1

该问题不影响 DDS 的正常通信，如需屏蔽该警告日志需要 root 权限启动。2

3.6.6 XML 文件解析失败 3

3.6.6.1 日志信息 4

Parse XML failed: %s. 5
使用”*_w_profile”接口时，解析 XML 配置失败。6

3.6.6.2 原因 7

Qos 配置文件不符合 XML 规范。8

3.6.6.3 解决方案 9

根据错误信息提示检查 XML 配置文件内容。10

3.6.7 零拷贝禁用可靠性策略 11

3.6.7.1 日志信息 12

Reliability has been disabled with ZeroCopyBytes writer %s of Topic %s. 13
在使用零拷贝数据通信时，DDS 自动禁用了数据可靠性策略。14

3.6.7.2 原因 15

使用零拷贝数据通信时，DDS 不会拷贝数据进行存储，无法对数据进行重传操作，可靠 16
性策略失效，所以 DDS 自动使用了尽力而为模式。
该警告日志不会影响正常数据通信。17

3.6.7.3 解决方案 18

使用零拷贝数据类型通信时，将可靠性配置设置为尽力而为模式。19

3.6.8 没有可用端口 1

3.6.8.1 日志信息 2

local(%s) no availiable port under domain(%d), release port or specific port and retry. 3
在指定域下没有可用端口。

3.6.8.2 原因 4

根据 DDS 协议规定，每个可用的端口数量是 250 个，每个 DP 会占用两个端口，同一节点上同一域号最多创建 124 个域参与者。当同一节点上同一域的参与者超过 124 时端口会超出范围，创建参与者失败。 5

3.6.8.3 解决方案 6

一般情况下，一个应用内在一个域下只需要使用一个域参与者，应用内共用域参与者，减少域参与者数量。或者将域参与者分配到不同的域下或不同的节点上。 7

4 排故流程 8

1) 确认使用环境（计算机环境、编译环境，网络环境）；9
2) 确认版本（如何获取版本号（以及使用编译语言），根据第一条日志判断版本号）；10
3) 保留日志文件，是否有报错日志以及警告，根据日志分析；11
4) 确认稳定复现场景（节点数量，数据流向，操作流程）；12
5) 在以前故障中匹配是否有类似的故障；13
6) 记录问题情况，进行分析与讨论，尝试复现；14
7) 记录问题解决方案，举一反三思考其他可能问题；15
8) 生成修复版本；16
9) 形成问题归零报告。17

附录 1socket 编程常见错误码及含义[1]

c Name	Value	Description	含义
EINTR	1	Interrupted system call	中断的系统调用
EBADF	2	Bad file number	无效文件描述符
EWOULDBLOCK	4	Same as “EAGAIN”	与 EAGAIN 的含义类似
EAGAIN	5	Try again	再次尝试
EMSGSIZE	7	Message too long	消息太长
EADDRINUSE	8	Address already in use	地址已被使用
ENETUNREACH	10	Network is unreachable	网络不可达
ECONNRESET	14	Connection reset by	连接被重置
ENOBUFS	15	No buffer space available	没有可用的缓存空间
EISCONN	16	Transport endpoint is already connected	传输端点已连接
ENOTCONN	17	Transport endpoint is not connected	传输端点未连接
ETIMEDOUT	18	Connection timed out	连接超时
ECONNREFUSED	20	Connection refused	连接被拒绝
EINPROGRESS	22	Operation now in progress	进程中正在进行的操作
EALREADY	23	Operation already in progress	操作已在进程中

[2]

EINVAL	26	Ivaild argument	无效参数
EMFILE	27	Too many open files	打开的文件过多
ENOTSOCK	28	Socket operation on non-socket	在非套接字上进行套接字操作
EDESTADDRREQ	29	Destination address required	请求目的地址
EOPNOTSUPP	34	Operation not supported on transport endpoint	操作上不支持传输端点
EAFNOSUPPORT	36	Address family not supported by protocol	协议不支持地址群
EADDRNOTAVAIL	37	Cannot assign requested address	无法分配请求的地址
EPERM	56	Operation not permitted	操作不允许
EIO	59	I/O error	I/0 错误
EPIPE	77	Broken pipe	管道破裂