ROS2 Fine Tunning

ROSCON 2017, Vancouver

Jaime Martin Losa
CEO eProsima
JaimeMartin@eProsima.com
+34 607 91 37 45

www.eProsima.com

Index

- ROS2 & XML profiles

Use cases (settings for Fast RTPS):

- Large Data & Video Streaming
- Low Bandwidth
- Deterministic Reliability
- Data Persistence
- Discovery Settings

ROS2 & XML profiles

ROS2 & XML profiles

- ROS2 creates a wrapper on top of the DDS middleware.
  - Simpler and convenient for the final user.
  - Hides fine configuration parameters.
- What if we need to fine tuning ROS2?
  - XML Profiles
    - simple mechanism to setup DDS entities
  - You need deep knowledge about DDS.
    - Fortunately we are publishing the settings for most interesting cases.
      - Besides this presentation, we will publish an article in our website with the example profiles

XML profiles

Fast RTPS 1.5.0 incorporates the ability to configure RTPS entities through XML profiles.

```
<?xml version="1.0" encoding="UTF-8" ?>
<profiles>
        <participant profile_name="participant_profile">
    ...
        </participant>

    <publisher profile_name="publisher_profile">
    …
    </publisher>

    <subscriber profile_name="subscriber_profile">
    …
    </subscriber>
</profiles>
```

XML profiles

- User can load XML files from code

  xmlparser::XMLProfileParser::loadXMLFile(“file.xml”);

and apply an XML profile to a entity

  Domain::createParticipant(“participant_profile”);

- **Also Fast RTPS is able to load a default XML file if it is found in execution directory:**

  DEFAULT_FASTRTPS_PROFILES.xml

And this is the mechanism we will use to setup ROS2

XML Profiles

Important Note: Some changes are required in ROS2 in order to fully use XML Profiles:

- **Right now ROS2 modifies some QoS in the RMW code, overwriting some of the values loaded by any provided XML file.**
- **We will publish next week a pull request to modify this behaviour.**

Large Data & Video Streaming

Large Data

- Fragmentation: DDS/RTPS uses UDP with a maximum message size of 64k
  - When you use Large data the middleware needs to fragment the messages

Large Data

- Fragmentation
  - Asynchronous publication mode:

```
<publishMode>
    <kind>ASYNCHRONOUS</kind>
</publishMode>
```

- Already by default.

In Asynchronous mode the user thread is not the responsible of the write operation. Another thread fragments the message and sends the fragments in the background

Video Streaming

- Fragmentation
- High throughput
- Streaming

Video Streaming

- High throughput
  - Best effort

```
<reliability>
    <kind>BEST_EFFORT</kind>
</reliability>
```

- It is ok to lose a few samples, as we are continuously sending new information.
- We avoid the HB, ACK/NACK traffic.

Video Streaming

- Streaming
  - Undefined stream length:
    <historyMemoryPolicy>PREALLOCATED_WITH_REALLOC</historyMemoryPolicy>
  - Focus in last frame:
    <historyQos>
      <kind>KEEP_LAST</kind>
      <depth>1</depth>
    </historyQos>

Video Streaming in lossy conditions

- Fragmentation
- High throughput
- Streaming
- Reliable communications

Video Streaming in lossy conditions

- Reliable communications
  - Reliability

```
<reliability>
    <kind>RELIABLE</kind>
</reliability>
```

  - Fast response by writer:

```
<times>
    <nackResponseDelay>
        <durationbyname>ZERO</durationbyname>
    </nackResponseDelay>
    <nackSupressionDuration>
        <durationbyname>ZERO</durationbyname>
    </nackSupressionDuration>
</times>
```

Video Streaming in lossy conditions

- Reliable communications
  - Fast response by reader:

```
<times>
    <heartbeatResponseDelay>
        <durationbyname>ZERO</durationbyname>
    </heartbeatResponseDelay>
</times>
```

Smart usage of network: avoiding bursts

- Fragmentation
- Flowcontrollers: Control the flow. You define how many fragments the middleware should send per a time period , avoiding sending all right away (a burst).
  - A burst usually leads to very low performance, collapsing the network.

Deterministic usage of network

- Flowcontrollers
  - Throughput controller

```
<throughputController>
    <bytesPerPeriod>1048575</bytesPerPeriod> <!-->1 Mbyte<-->
    <periodMillisecs>1000</periodMillisecs>
</throughputController>
```

Low Bandwidth

Low Bandwidth

- Reduce bandwidth usage
  - Use case example: VHF Radio, 4800 bits per second.

Low Bandwidth

- Reduce bandwidth usage
  - Static discovery: See Fast RTPS user manual

```
<builtin>
    <EDP>STATIC</EDP>
    <staticEndpointXMLFilename>StaticDiscoveryConf.xml
    </staticEndpointXMLFilename>
</builtin>
<name>ParticipantWithPublisher</name>
```

Automatic discovery in DDS is very chatty, you need the order **n** squared messages being n the number of nodes. To avoid this we can use static discovery, a simplified mechanism to load some discovery data from local xml files. This mechanism uses a number of messages proportional to n.

See the user manual for further explanation

Low Bandwidth

- Reduce bandwidth usage
  - Data over multicast, add to the readers:

```
<multicastLocatorList>
    <locator>
        <address>239.255.0.4</address>
        <port>7600</port>
    </locator>
</multicastLocatorList>
```

Low Bandwidth

- Reduce bandwidth usage
  - Increase participant announcement period

```
<builtin>
   <leaseDuration>
      <durationbyval>
         <seconds>1800</seconds>
      </durationbyval>
   </leaseDuration>
   <leaseAnnouncement>
      <durationbyval>
         <seconds>360</seconds>
      </durationbyval>
   </leaseAnnouncement>
</builtin>
```

Low Bandwidth

- Reduce bandwidth usage
  - Reduce writer's metatraffic data

```
<heartbeatPeriod>
    <durationbyval>
        <seconds>180</seconds>
    </durationbyval>
</heartbeatPeriod>
<nackResponseDelay>
    <durationbyval>
        <seconds>30</seconds>
    </durationbyval>
</nackResponseDelay>
<nackSupressionDuration>
    <durationbyval>
        <seconds>30</seconds>
    </durationbyval>
</nackSupressionDuration>
```

Low Bandwidth

- Reduce bandwidth usage
  - Reduce reader's metatraffic data

```
<heartbeatResponseDelay>
    <durationbyval>
        <seconds>1</seconds>
    </durationbyval>
</heartbeatResponseDelay>
```

Deterministic Reliability

Deterministic Reliability

- Reliable communications
- Avoid losing any sample

Deterministic Reliability

- Reliable communications
  - Reliability

```
<reliability>
    <kind>RELIABLE</kind>
</reliability>
```

- Avoid losing any sample
  - Store all samples in both sides

```
<historyQos>
    <kind>KEEP_ALL</kind>
</historyQos>
```

Sending commands with low latency

- Reliable communications
- Avoid losing any sample
- Maximum Determinism
  - Get repair messages inmediatly if any sample is lost.

Deterministic Reliability

- Maximum Determinism
  - Fast heartbeat period

```
<heartbeatPeriod>
    <durationbyval>
        <fraction>429496700</fraction> <!-- 100 ms -->
    </durationbyval>
</heartbeatPeriod>
```

Deterministic Reliability

- Maximum Determinism
  - Fast response by writer:

```
<times>
    <nackResponseDelay>
        <durationbyname>ZERO</durationbyname>
    </nackResponseDelay>
    <nackSupressionDuration>
        <durationbyname>ZERO</durationbyname>
    </nackSupressionDuration>
</times>
```

Deterministic Reliability

- Maximum Determinism
  - Fast response by reader:

```
<times>
    <heartbeatResponseDelay>
        <durationbyname>ZERO</durationbyname>
    </heartbeatResponseDelay>
</times>
```

Data Persistence

Sending already published data to late joiners

Data Persistence

- Reliable communications
- History
- Persistence

Data Persistence

- Reliable communications
  - Reliability

```
<reliability>
    <kind>RELIABLE</kind>
</reliability>
```

- History
  - How many samples will be stored

```
<historyQos>
    <kind>KEEP_LAST</kind>
    <depth>100</depth>
</historyQos>
```

Data Persistence

- Persistence
  - Make history available for late joiners.

```
<durability>
    <kind>TRANSIENT_LOCAL</kind>
</durability>
```

Discovery Settings

Discovery: Why could I change this?

- By default, discovery mechanism uses multicast messages for participant discovery.
  - Sometimes multicast is not available, or you want to use different multicast addresses.

Discovery Settings

- Initial peers
- Metatraffic multicast locators
- Metatraffic unicast locators

Discovery Settings

- Initial peers
  - Location of remote participants

```
<initialPeersList>
    <locator>
        <address>192.168.1.3</address>
        <port>7410</port>
    </locator>
</initialPeersList>
```

Discovery Settings

- Metatraffic multicast locators
  - Where they can send me metatraffic data using multicast

```
<metatrafficMulticastLocatorList>
    <locator>
        <address>239.255.0.1</address>
        <port>7400</port>
    </locator>
</metatrafficMulticastLocatorList>
```

Discovery Settings

- Metatraffic unicast locators
  - Where they can send me metatraffic data using unicast

```
<metatrafficUnicastLocatorList>
    <locator>
        <address>192.168.1.2</address>
        <port>7410</port>
    </locator>
</metatrafficUnicastLocatorList>
```

Want to know more?

www.eProsima.com

Youtube:
https://www.youtube.com/user/eprosima

Mail: JaimeMartin@eProsima.com
Phone: +34 607913745

Twitter: @jaimemartinlosa
http://es.slideshare.net/JaimeMartin-eProsima

Thank you