ROS2 and DDS
St. Louis C++ Meetup

Jeremy Adams and Adam Mitz
jeremy@allegrobotllc.com,
mitza@objectcomputing.com
September 11, 2019

Outline

- ROS2 Introduction
- DDS Introduction
- ROS2 Motivations
- ROS2 Highlighted Features
- Developing DDS Applications
- DDS Wire Protocol (RTPS)
- DDS Implementations & ROS2 RMW
- References

ROS2: Video Intro

ROS 1 developed as more of a research platform but has become very popular

ROS2: Robots

Supported Robots

- Aerial
- Components
- Ground
- Manipulator
- Marine

What is DDS?

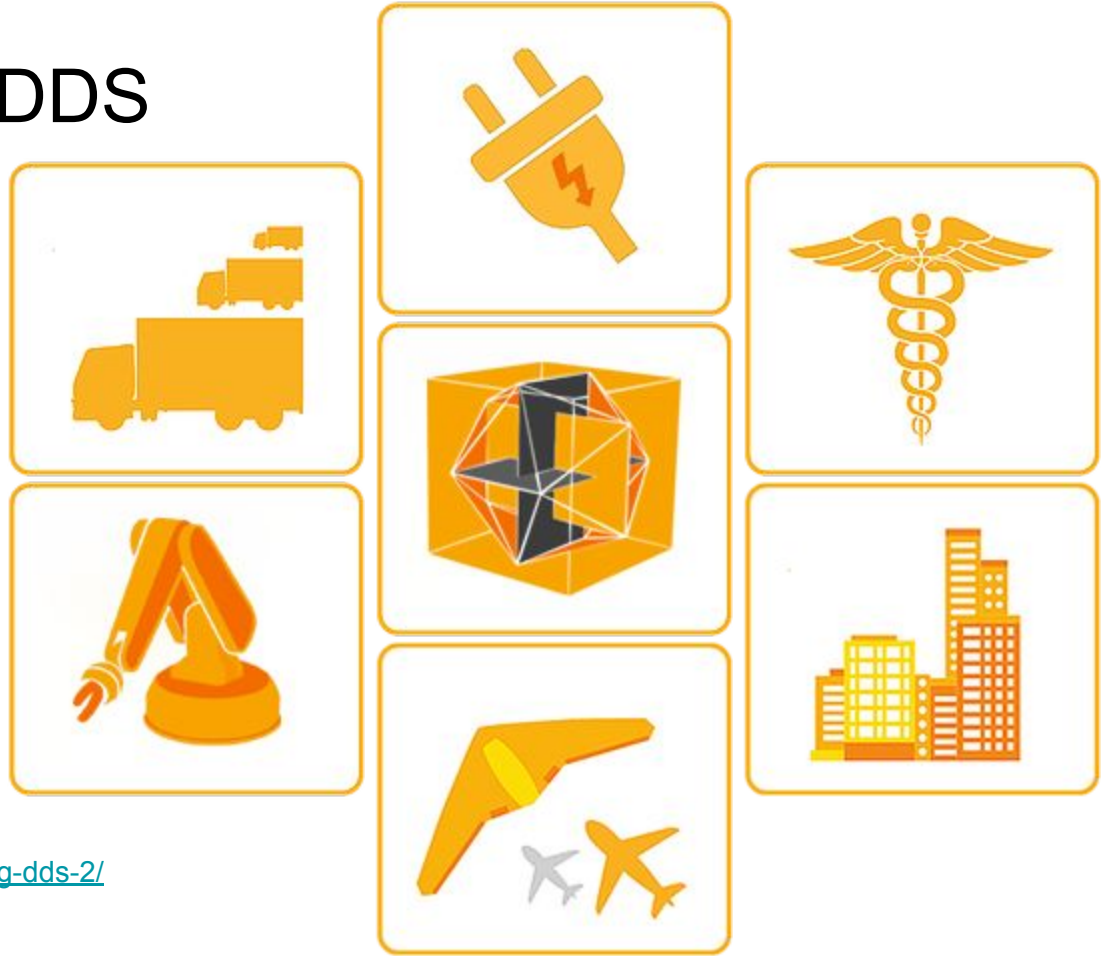
The Data Distribution Service (DDS™) is a middleware protocol and API standard for data-centric connectivity. It integrates the components of a system, providing low-latency data connectivity, extreme reliability, and a scalable architecture that business and mission-critical Internet of Things (IoT) applications need.

In a distributed system, middleware is the software layer that lies between the operating system and applications. It enables the various components of a system to more easily communicate and share data. It simplifies the development of distributed systems by letting software developers focus on the specific purpose of their applications rather than the mechanics of passing information between applications and systems.

https://www.dds-foundation.org/what-is-dds-3/

Industries Using DDS

- Transportation
- Smart Energy
- Medical Devices
- Industrial Automation
- Simulation / Test
- Smart Cities
- Military / Aerospace



https://www.dds-foundation.org/who-is-using-dds-2/

DDS Compared to Other Pub/Sub Technology

Data Distribution Service (DDS)

- Data-Centric
  - Distributed global state of data
- Footprint Friendly
  - Enterprise to Micro
- Real-time Processing
- Decentralized Discovery/Queuing
  - Plug-n-Play
  - In-process caching/queuing
  - Architecturally De-coupled
- Rich Quality of Service to match distribution/timing/resiliency/security
- Native-language Data Object definition and runtime instances

Other Pub/Sub Solutions

- Message Oriented
  - Passing payload of non-globally defined data
- Traditionally enterprise and less in real-time
- Centralized Discovery/Queuing
  - Daemon-based discovery
  - Agent-based queuing
  - Increased dependency management and potential points of failure
  - Architecturally Coupled
- Increased transformation logic for opaquely defined message payloads

ROS2: Why ROS2?

- Compute spread across robots and embedded devices in the system
- Take advantage of Real-time embedded
- Operate in lossy networks
- Migrate from tools for academia to tools for creating product
- Support all major OSs (Linux, Mac, and Windows)
- Add support for security mechanisms

http://design.ros2.org/articles/why_ros2.html

ROS2: motivations

- Standards-based message transport - DDS
  - Enables communication with other devices using the standard
- Currently C++14
  - plans to move forward in future releases
- Robot distributed computing
- Real-time support
- Low quality network support (DDS/RTPS helps with this)
- Service discovery to quickly find resources via topics
- Node lifecycle (Node/process state)
- Security built-in, based mostly on DDS Security
- Launch in Python enabling more complex logic than ROS1 launch

ROS2: motivations (cont.)

- Dynamic loading of nodes in composition
- Support for linux, Mac, and Windows
- C++ and Python standard language bindings but others (C#, Rust, Go, etc.) supported through community support

http://design.ros2.org/articles/why_ros2.html

ROS2: Highlighted Features

- Introspection
- Composition
- Efficient Intra-process (non DDS) communications
- Node lifecycle management
- QoS using DDS

https://github.com/ros2/demos/tree/0.7.8

ROS2: Introspection

- Tools
  - Show all nodes
    - `ros2 node list`
  - Show all topics
    - `ros2 topic list`
  - Echo topic
    - `ros2 topic echo /chatter`
  - Publish topic from command line
    - `ros2 topic pub …`
  - Publish topic from command line
    - `ros2 run rqt_gui rqt_gui`

https://index.ros.org/doc/ros2/Tutorials/Introspection-with-command-line-tools/

ROS2: Composition

- Typical use case is to run multiple nodes in a single process
  - Dynamic composition
    - Component container
    - dlopen
  - Manual composition
  - Using launch
- Dynamic loading allows for more of a plug-in type node
  - Startup speed - only load what's needed at startup
  - Runtime implementation selection
  - Nodes optimized for hardware configuration (CPU, GPU, etc.)

https://index.ros.org/doc/ros2/Tutorials/Composition/

ROS2: Efficient Intra-process (non DDS)

- Using zero copy interface instead of DDS
- Can still show nodes
- Can still echo topics
- Similar to ROS1 nodelets

https://index.ros.org/doc/ros2/Tutorials/Intra-Process-Communication/

ROS2: Node lifecycle management

- Change lifecycle states using ROS2 services
- Primary States (steady states):
  - unconfigured
  - inactive
  - active
  - shutdown
- Transition States (intermediate states):
  - configuring
  - activating
  - deactivating
  - cleaningup
  - shuttingdown

https://github.com/ros2/demos/blob/master/lifecycle/README.rst

ROS2: QoS using DDS

- QoS policies
  - History
    - Keep last N samples
    - Keep all
  - Depth
    - Size of queue
  - Reliability
    - Best effort - telemetry
    - Reliable - command and control
  - Durability
    - Data Writer persists messages until datareader is available

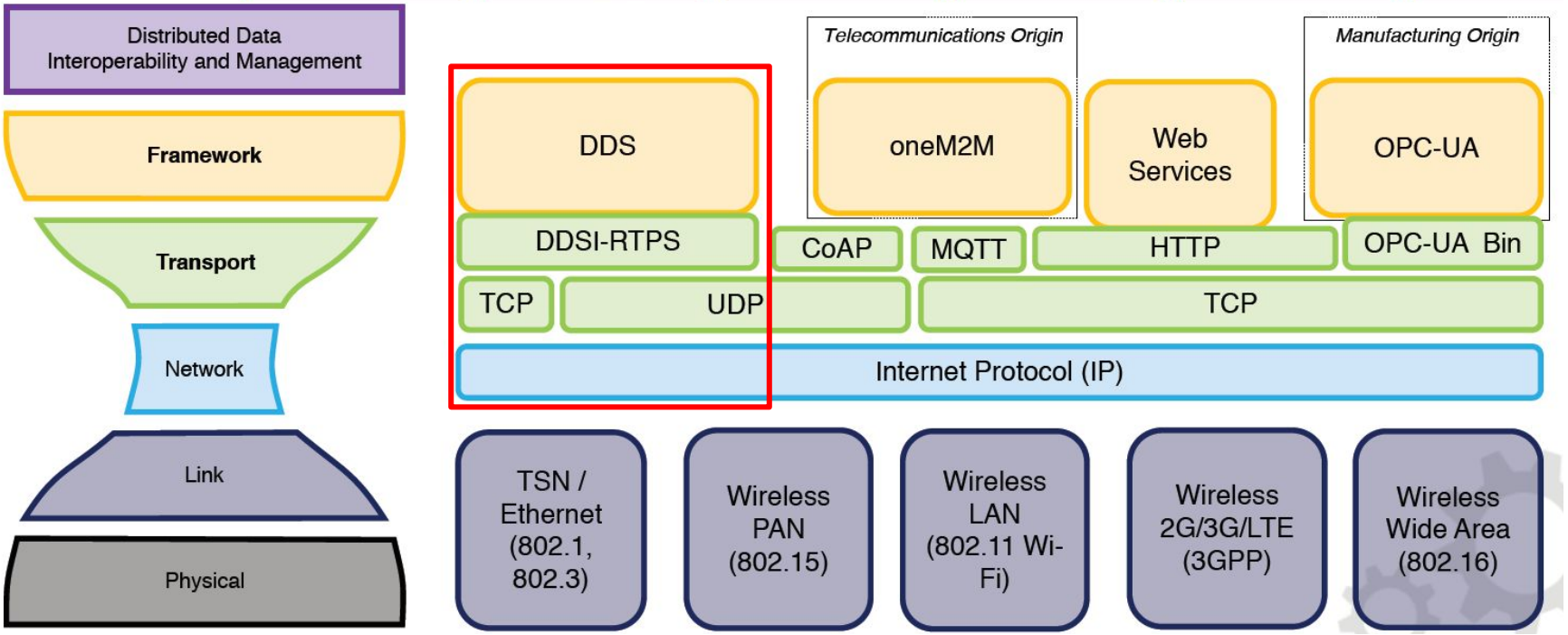
https://design.ros2.org/articles/qos.html

ROS2: QoS using DDS

- Focus on reliability QoS
  - Reliable
  - Best effort

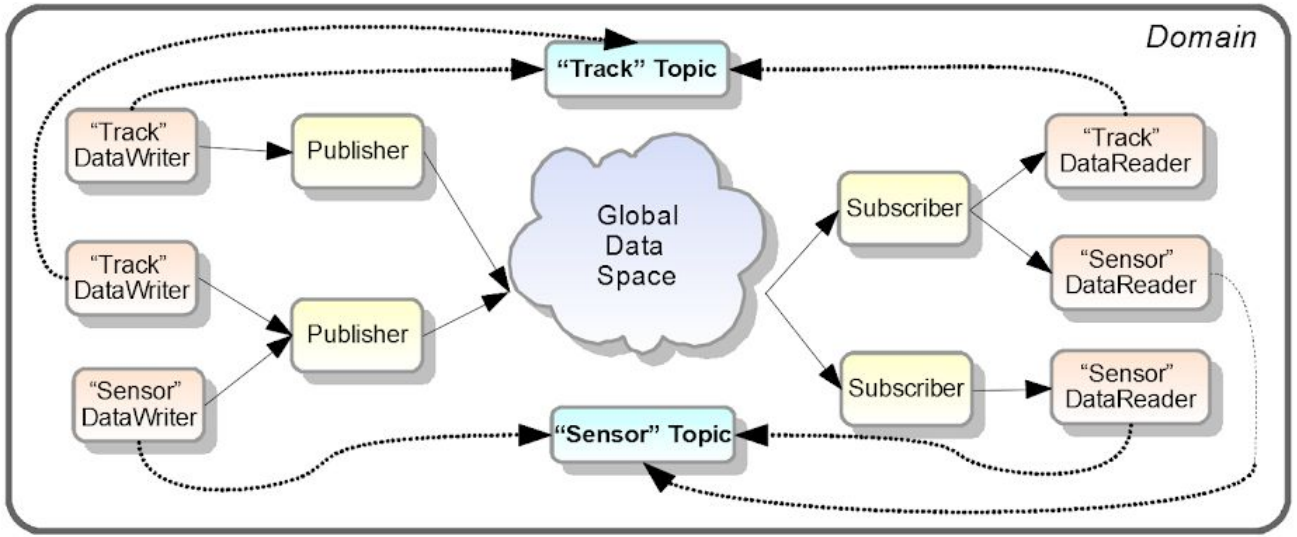
https://index.ros.org/doc/ros2/Tutorials/Quality-of-Service/

DDS: Framework and Transport



Developing DDS Applications: Overview

- Participants, Topics, Data Writers, Data Readers
- Data types for each Topic (uses IDL)
- QoS policies
- DDS Security, other config / deployment considerations




```
struct Quote
{
  string ticker;
  string exchange;
  string full_name;
  double value;
};
```

Developing DDS Applications: Example Publisher

```
StockQuoter::Quote rht_quote;
rht_quote.exchange = STOCK_EXCHANGE_NAME;
rht_quote.ticker = "RHT";
rht_quote.full_name = "Red Hat";
rht_quote.value = 16.0 + 0.5 * i;
const auto ret = quote_dw->write(rht_quote, DDS::HANDLE_NIL);
```

Developing DDS Applications: Example Subscriber

```
StockQuoter::Quote quote;
DDS::SampleInfo si;
while (quote_dr->take_next_sample(quote, si) == DDS::RETCODE_OK) {
  if (si.valid_data) {
    cout << "Quote: ticker    = " << quote.ticker << endl
         << "       exchange  = " << quote.exchange << endl
         << "       full name = " << quote.full_name << endl
         << "       value     = " << quote.value << endl;
  } else if (si.instance_state ==
              DDS::NOT_ALIVE_DISPOSED_INSTANCE_STATE) {
      cout << "Quote DISPOSED " << quote.ticker << endl;
  }
}
```

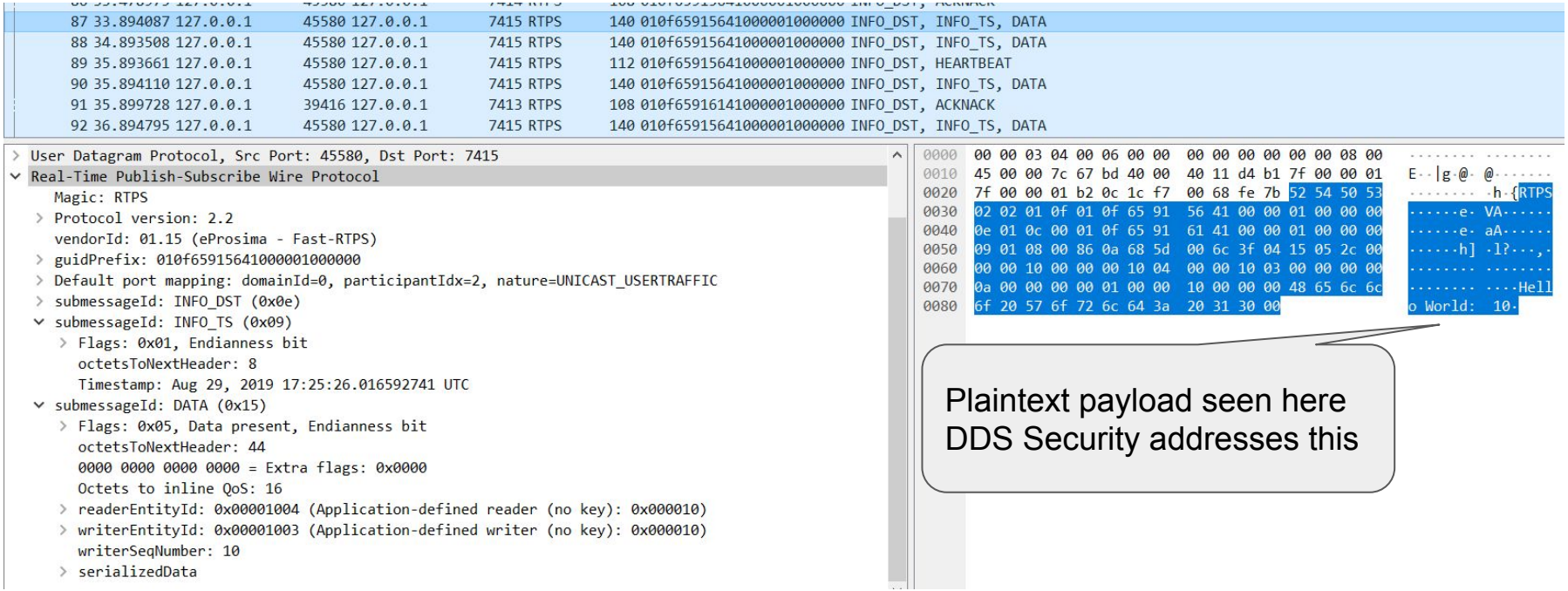
The DDS Middleware Specification

- Defines common terms, APIs, behaviors
- Use of typed data, instance lifecycle
- DDS "Entities" (DomainParticipant, DataWriter, DataReader, Topic, …)
- Distributed shared cache model
- Quality of Service Policies (Presentation, Deadline, Ownership, Partition, …)
- Content-subscription features
- Responding to events: Listeners or Conditions
- Accessing to data about the domain: Built-in Topics

The RTPS Wire Protocol (DDSI-RTPS)

- DDS standard transport: Real-Time Publish-Subscribe (RTPS) protocol
- Runs on top of UDP/IP
  - Could also use other underlying protocols
  - UDP/IP is currently standardized; TCP/IP in work
- Supports multicast, doesn't require it
- Content-aware reliability
- Peer-to-peer dynamic discovery, direct data transmission (no broker)
- DDS-Security (optional add-on spec)
  - Diffie-Hellman Key Exchange
  - Access Control permissions model
  - AES-GCM 256-bit Encryption
  - Works with multicast, sender can encrypt once for multiple recipients

Wireshark packet capture: ROS2 with Fast RTPS



DDS: OpenDDS and other DDSs

Specifies

Implemented By

Developing In

Produces

Completely Open Source.
Implements DDS, RTPS,
DDS-Security specs.

Other Implementations:

Provides the Fast RTPS
framework, isn't a DDS

Differences include support for optional specs, license models, programming language bindings

ROS2 "Dashing" Release: RMW Implementations

Product name	License	RMW implementation	Status
eProsima *Fast RTPS*	Apache 2	`rmw_fastrtps_cpp`	**Full support**. Default RMW. Packaged with binary releases.
RTI *Connext*	commercial, research	`rmw_connext_cpp`	**Full support**. Support included in binaries, but Connext installed separately.
RTI *Connext* (dynamic implementation)	commercial, research	`rmw_connext_dynamic_cpp`	Support paused. Full support until alpha 8.*
ADLINK *Opensplice*	Apache 2, commercial	`rmw_opensplice_cpp`	Partial support. Support included in binaries, but OpenSplice installed separately.
OSRF *FreeRTPS*	Apache 2	–	Partial support. Development paused.

"Partial support" means that one or more of the features required by the rmw interface is not implemented.

https://index.ros.org//doc/ros2/Concepts/DDS-and-ROS-middleware-implementations/ (emphasis added)

https://www.ros.org/reps/rep-2000.html#dashing-diademata-may-2019-may-2021

References

- ROS2
  - index.ros.org/doc/ros2/Releases/Release-Dashing-Diademata
- ROS Industrial
  - rosindustrial.org
- DDS Foundation
  - dds-foundation.org
- OpenDDS
  - opendds.org
  - github.com/objectcomputing/OpenDDS
  - objectcomputing.com/products/opendds