Vol.2018-EMB-48 No.2
2018/6/29

# ROS2と軽量DDSの組込みシステムに対する適用性評価

小澤 慶祐[1]　本田　晋也[1]　松原 豊[1]　高田 広章[1]　加藤 寿和[2]　山本 整[2]

**概要**：近年，サービスロボットや自動運転のような自律制御システムの研究開発により，組込みシステムのアーキテクチャが著しく変化している．本研究では，ROS2 と軽量 DDS による通信が混在する自律制御システムを想定し，そのシステムの満たすべき要件を設定した．そして，ROS2 と軽量 DDS がそれらの要件を満たすか評価するための項目を設定し，適用性を評価した．評価実験では，主に自律制御システムの要求するリアルタイム性を満足できるか評価するために，ROS2 および軽量 DDS の通信時間を計測した．評価実験を行った結果，ROS2 は通信時間が大きく，特に通信以外の処理のオーバヘッドが大きいことがわかった．また，通信時間のばらつきも大きいことがわかり，自律制御システムに適用するには課題が残っていると考えられる．一方，軽量 DDS は ROS2 に比べて，通信時間が小さく，通信時間のばらつきも小さいという結果が得られ，組込みシステムへの適用が現実的だと考えられる．

**キーワード**：ROS2, DDS， 組込みシステム， 自律制御システム， ソフトウェアプラットフォーム

# Evaluation of applicability of ROS2 and lightweight DDS to embedded system

Keisuke Kozawa[1]　Shinya Honda[1]　Yutaka Matsubara[1]　Hiroaki Takada[1]　Toshikazu Kato[2]
Hitoshi Yamamoto[2]

***Abstract:*** In recent years, the architecture of an embedded system has developed dramatically due to research and development of autonomous control system such as service robot and automated driving. In this research, we assumed an autonomous control system with mixed ROS2 and lightweight DDS communication, and set up requirements to be satisfied by the system. We set items for evaluating whether ROS2 and lightweight DDS satisfy these requirements, and evaluated its applicability. In the evaluation experiment, the communication time of ROS 2 and lightweight DDS was measured mainly to evaluate whether real time property required by autonomous control system can be satisfied. As a result of the evaluation experiment, it was found that the communication time of ROS2 is large, especially the overhead of processing other than communication is large. Moreover, it is found that the dispersion of the communication time is large, and it seems that the problem remains to be applied to the autonomous control system. On the other hand, lightweight DDS has smaller communication time and less communication time variation than ROS2, and applying to embedded systems is considered to be realistic.

***Keywords:*** ROS2, DDS, an embedded system, autonomous control system, software platform

## 1. はじめに

近年，サービスロボットや自動運転のような自律制御システムの研究開発により，組込みシステムのアーキテクチャが著しく変化している．現在，自律制御システムのアプリケーションのプロトタイプ開発においては，モジュール間の通信にロボット制御ミドルウェア ROS（Robot Operating System）が使用されているケースが多い．ROS はライブラリやシミュレーション環境が充実しており，開発段階で使用することに適しているが，リアルタイム性や信頼性が保証されておらず，実際の自律制御システムへの適用には課題がある [1]．そこで，これらの問題を解決することを目的として，ROS2 が開発された．

ROS2 は通信に DDS（Data Distribution Service）を使用することで ROS の問題点を解決しようとしている [2][3]．自律制御システムにおいて，ROS2 は UNIX 系 OS が動作する高性能な CPU 上で動作し，認識や判断に関わる高い処理性能を必要とするアプリケーションを実行することを

[1] 名古屋大学 大学院情報学研究科
[2] 三菱電機株式会社 情報技術総合研究所

想定している．しかし，システムは複数の CPU によって構成されており，全ての CPU で高性能な CPU を用いることはコストの面で難しい．よって，高い処理性能を必要としないアクチュエータやセンサを制御するアプリケーションは，低性能な CPU を用いる必要がある．このような低性能 CPU では，リソースが小さく ROS が要求するライブラリを持たないため，ROS2 を動作させることは困難である．そこで，組込みシステム向けに開発された軽量 DDS を用いることが現実的である．

ROS2 は 2017 年 12 月に正式にリリースされており，評価が十分に行われていない．同様に，軽量 DDS も評価が十分に行われていない．そこで，本研究では ROS2 と軽量 DDS が混在する自律制御システムを想定し，そのシステムが満たすべき要件を設定した．そして，ROS2 と軽量 DDS がそれらの要件を満たすか評価するための項目を設定し，評価実験を行うことで，適用性を評価した．

ROS2 は高性能 CPU として Raspberry Pi 3 上で，軽量 DDS は低性能 CPU として STM32F4 Discovery 上で動作させ評価実験を行った [4][5]．また，ROS2 の DDS としては ROS2 のデフォルトの DDS 実装である FastRTPS を，軽量 DDS としては ROS を管理する OSRF（Open Source Robotics Foundation）が開発した FreeRTPS を使用した [6][7]．評価実験では，大きく分けて以下の 3 つの実験を行った．1 つ目に自律制御システムのリアルタイム性を満足できるか評価するために，ROS2 および軽量 DDS の通信時間を計測した．2 つ目に，自律制御システムにおいて，CPU は状況に応じて電源の ON と OFF を繰り返し，アプリケーションが必要なときだけに動作することが想定されるため，ROS2 および軽量 DDS の初期化に要する時間を計測した．最後に，リソース制約の厳しい組込みシステムに適用できるかを評価するために，軽量 DDS のメモリ使用量を計測した．

## 2. ROS2 と軽量 DDS

本章では，ROS2 と軽量 DDS について述べる．

### 2.1 ROS2

ROS2 は，リアルタイムなタスクや信頼性のある通信に対応できるように ROS とは別に開発された．

ROS2 は通信層に DDS を使用している．DDS は publish/subscribe 通信を提供する国際ミドルウェア標準であり，ROS2 は DDS を使用して，プロセス間通信を行っている [2]．

ROS2 のソフトウェア構成は主に rcl（ROS client library）と rmw（ROS middleware）に分かれている．ROS2 のソフトウェア構成を図 1 に示す．

ROS2 が使用している DDS はさまざまなベンダによって実装されている．rmw はユーザアプリケーションまたは rcl が DDS を使用するために用意された API 群であり，DDS の実装ごとに提供されている．ROS2 ではユースケースに合わせて適した DDS の実装を使用できるように，複数の rmw を提供している [8]．

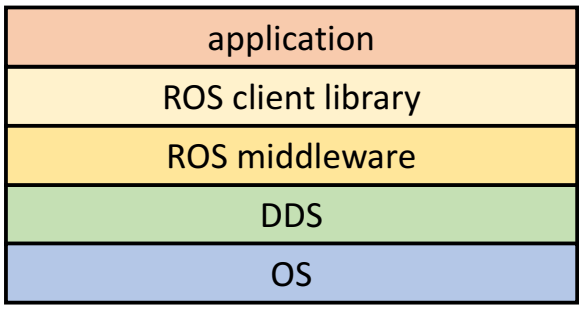



図 1　ROS2 のソフトウェア構成

ROS2 には QoS (Quality of Service) ポリシーと呼ばれる独自の QoS が用意されており，QoS ポリシーの組み合わせにより QoS プロファイルを形成する．また，ROS2 では用途に合わせた QoS プロファイルのプリセットが用意されており，容易に QoS ポリシーを設定することができる．ROS2 が提供する基本 QoS プロファイルに含まれている QoS ポリシーを以下に示す [9]．

- History
  - Keep last: Depth で指定した数だけ送信待ちメッセージをキューに格納する．
  - Keep all: ミドルウェアのリソースの上限まで送信待ちのメッセージをキューに格納する．
- Depth
  - Size of the queue: Keep last を設定したときのキューに送信待ちのメッセージを格納する数を設定する．
- Reliability
  - Best effort: メッセージの送信を失敗した場合でも，メッセージを一度しか送信しない．
  - Reliable: メッセージの送信を失敗した場合に，メッセージを再送する．
- Durability
  - Transient local: 後から作成された subscriber に送信するために，publisher が古いメッセージを保存する．
  - Volatile: publisher が古いメッセージを保存しない．

### 2.2 DDS

DDS とは，OMG（Object Management Group）によって策定されたリアルタイムおよび組込みシステム向けに publish/subscribe 通信を提供するオープンな国際ミドルウェア標準である [3]．DDS は輸送管理システムや医療機器，産業オートメーション，航空宇宙システムなどの採用実績がある．

DDS は publish/subscribe 通信を提供する．DDS の特

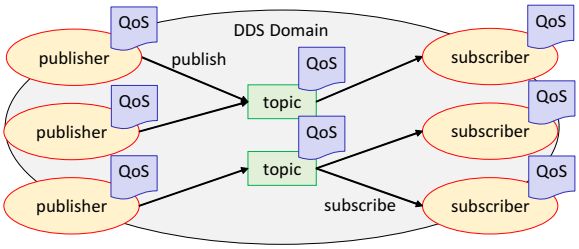



図 2　DDS の通信モデル

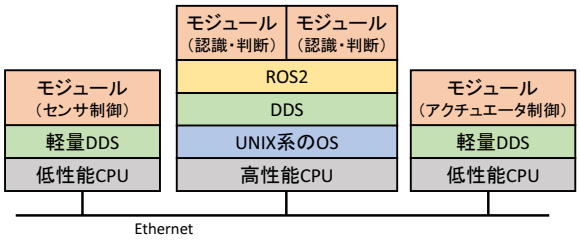



図 3　対象システムの構成図

徴として QoS と呼ばれる，システムの動作を制御するパラメータを設定することで，リソースの消費量や，フォールトトレランス，通信の信頼性などを設定できることがあげられる．この QoS は publisher や subscriber，トピック毎に設定が可能である．DDS の提供する publish/subscribe 通信の通信モデルを図 2 に示す．

また本研究では，DDS の仕様を完全にサポートしておらず，DDS の通信を行うことができる範囲で機能を限定したものを軽量 DDS と呼んでいる．軽量 DDS として，OSRF が開発した FreeRTPS を評価実験に使用した．他の軽量 DDS としては RTI が開発した Connext DDS Micro などがあげられる．

## 3. 関連研究

ROS2 の評価に関する研究としては，文献 [10] で PC 上での ROS2 の評価に関する研究が行われている．ROS2 の評価として，ROS との性能の比較，ROS2 で使用する DDS の実装によっての性能の比較を行っている．また，DDS の QoS の設定を変更しての性能の比較も行っている．そして，ROS2 が ROS より通信の信頼性が高いといった結果が得られている．また，DDS の実装によって機能や性能に違いがあることがわかっている．

ROS が無線アドホックネットワークで使用される場合，帯域幅が限られることや遅延，ジッタなどの問題が存在する．文献 [11] では，Pound と呼ばれる ROS ノードを提案している．提案手法ではコントロールオーバネットワークの例を含め，すべてのテストケースにおいて，他の最先端の手法と同等かそれ以上の性能を発揮することがわかっている．

現在 ROS2 では，システム毎に 1 つの DDS 実装しか指定できない．文献 [12] では，特徴の異なる通信毎に最適な DDS 実装を動的に切り替える手法を提案している．提案手法では，通信開始時にデータサイズや通信範囲などの通信特性を推定し　最適な DDS 実装を自動的に選択することで，現在の ROS2 と比較して有意性を示している．

これらの評価は，PC を用いており，組込みシステムを対象とした評価はなされていない．

## 4. 適用性評価の対象システムと評価項目

本章では適用性を評価するために，対象システムの構成及び要件を設定し，そこから評価項目を設定する．

### 4.1 システム構成

本研究では自律制御システムのモジュール間通信に ROS2 および軽量 DDS を使用することを想定した．自律制御システムは複数の CPU によって構成され，さまざまな機能を持ったモジュールが各 CPU に割り当てられる．自律制御システムでは，認識や判断を行うモジュールと，モータなどのアクチュエータを制御するモジュール，周囲の情報を取り入れるセンサを制御するモジュールが主なモジュールとしてあげられる．認識や判断を行うモジュールは高い処理性能を必要とし，高性能 CPU 上で動作することを想定した．しかし，システム上の全ての CPU で高性能 CPU を用いることはコストの面で難しい．よって，センサやアクチュエータを制御するような高い処理性能を必要としないモジュールは，低性能 CPU 上で動作することを想定した．そこで，高性能 CPU 上では UNIX 系の OS が動作し，モジュール間通信に ROS2 を使用する．一方，低性能 CPU では OS は動作せずにモジュール間通信に軽量 DDS を直接用いる．また，自律制御システムのネットワークは Ethernet で構成され，各マイコンが LAN によって接続され，ネットワーク速度 100Mbps であると想定した．以上の想定をした自律制御システムの構成図を図 3 に示す．

### 4.2 システム要件

**（要件 1）通信のリアルタイム要求**

自律制御システムでは，常に変化し続ける周囲の状況に確実に対応する必要がある．よって，各モジュール間の通信には高いリアルタイム性が要求される．また，システムが長時間動作し続けても通信時間が一定である必要がある．

**（要件 2）アプリケーションの起動速度要求**

組込みシステムでは，CPU は状況に応じて電源の ON と OFF を繰り返すため，アプリケーションが必要なときだけに起動し，動作することが想定される．よって，CPU の電源が ON に切り替わり，通信を開始するまでの時間がかかってしまうとリアルタイム性に問題があるため，通信

を開始するまでの時間が短いことが要求される．

**（要件 3）軽量 DDS のメモリ消費要求**

本研究では，組込みシステムにおいて，軽量 DDS は低性能なマイコン上で動作することを想定している．また，低性能なマイコンはメモリ量が多くないことが想定される．よって，軽量 DDS はメモリの使用量が少ないことが要求される．

### 4.3 評価項目

前節で設定した要件を評価するために設定した評価項目を以下に示す．

**（項目 1）ROS2 および軽量 DDS の通信時間**

ROS2 および軽量 DDS の通信時間を計測した．ROS2 および軽量 DDS の通信時間は以下のような通信の組み合わせで計測を行った．

- 1 台の CPU の ROS2 内のノード間の通信
  認識や判断を行う ECU 内であるアプリケーションと別のアプリケーションが通信を行う場合を想定している．
- 2 台の CPU の別の ROS2 間の通信
  認識や判断を行う CPU が複数個存在し，その CPU 間で通信を行う場合を想定している．
- ROS2 から軽量 DDS へメッセージを送る通信
  認識や判断を行う CPU がアクチュエータを制御する CPU に制御に必要な情報を送る場合を想定している．
- 軽量 DDS から ROS2 へメッセージを送る通信
  センサを制御する CPU から認識や判断を行う CPU にセンサで読み取った情報を送る場合を想定している．
- 2 台の CPU の別の軽量 DDS 間の通信
  センサを制御する CPU 間で，情報を統合する場合や，アクチュエータを制御する CPU 間で制御の連携をとる場合を想定している．

また，ROS2 は通信に UDP を使用しているため，高性能 CPU 内と高性能 CPU 間の UDP の通信時間を計測し，ROS2 の通信以外の処理にかかるオーバヘッドも求めた．

さらに，以下の条件で通信時間を計測した．

**（項目 1.1）時間経過による通信時間の変化**

計測を行っている過程で，通信を開始してから時間が経過すると通信時間が大きくなる傾向が見られたため，時間経過と通信時間の関係を調べるために計測を行った．

**（項目 1.2）メッセージサイズによる通信時間の変化**

サービスロボットや自動運転ではカメラから取り入れた映像など容量の大きいデータをやり取りする必要があるため，メッセージサイズと通信時間の関係を調べるために計測を行った．

**（項目 1.3）軽量 DDS のノード数による通信時間の変化**

認識や判断を行う CPU がセンサを制御する複数の CPU からデータを受け取ることや，アクチュエータを制御する複数の CPU にデータを送信することが想定される．そこで，ROS2 が軽量 DDS と通信する際に軽量 DDS のノード数を変化させ，ノード数と通信時間の関係を調べるために計測を行った．

**（項目 1.4）QoS の設定による通信時間の変化**

組込みシステムでは通信の内容や状況によって様々な設定の QoS を使用して通信を行うことが想定される．そこで，様々な設定の QoS を使用して通信時間の計測を行った．

**（項目 2）ROS2 および軽量 DDS の初期化に要する時間**

アプリケーションの初期化に要する時間を計測した．具体的にはアプリケーションが起動してから最初に通信を完了するまでの時間を計測した．

**（項目 3）軽量 DDS のメモリ使用量**

軽量 DDS のメモリ使用量を計測した．メモリの使用量としては，低性能 CPU に書き込むオブジェクトファイルの ROM および RAM の使用量を計測した．

## 5. 評価

本章では前章で設定した内容について，評価実験を行った手法とその結果，結果に対する考察を述べる．評価実験を行った評価環境の詳細を表 1 に示す．

### 5.1 評価手法

評価実験を行った評価手法の詳細を以下に示す．

**（項目 1）ROS2 および軽量 DDS の通信の通信時間**

ROS2 および軽量 DDS の通信時間を計測するために，publisher がメッセージを送信する API を呼ぶ直前から subscriber がメッセージを受信し，コールバック関数を呼び出すまでを計測し，この時間を通信時間とした．さらに，ROS2 の通信では UDP を使用しているため，高性能 CPU 内と高性能 CPU 間において UDP の通信時間も計測することにより，ROS2 の通信以外にかかるオーバヘッドを求めた．計測の条件としては，”Hello World”という文字列をメッセージとして 10Hz で送信し，100 秒間通信時間を計測した．また，計測を行ったタイミングは通信を開始した直後である．QoS の設定を変更して行った評価実験以外の評価実験で使用した QoS の設定を表 2 に示す．

**（項目 1.1）時間経過による通信時間の変化**

通信を開始した直後，15 分後，30 分後，45 分後，60 分後の 5 つのタイミングでそれぞれ通信時間を計測した．

**（項目 1.2）メッセージサイズによる通信時間の変化**

送信するメッセージを 64byte，128byte，256byte，512byte，1024byte の 5 つでそれぞれ通信時間を計測した．

**（項目 1.3）ノード数による通信時間の変化**

ROS2 と通信する軽量 DDS のノード数を 1 つから 4 つまで変化させて通信時間を計測した．また，複数の軽量 DDS が同じトピックで通信をして計測した．

表 1　各 CPU のスペック

| | 高性能 CPU | 低性能 CPU |
|---|---|---|
| ハードウェア | Raspberry Pi 3 Model B[4] | stm32F4 Discovery[5] + stm32F4DIS-BB[13] |
| 動作周波数 | 600MHz に固定 | 168MHz に固定 |
| ネットワーク速度 | 100Mbps | 100Mbps |
| コア数 | 4（アプリケーションごとにコアを固定） | 1 |
| OS | Ubuntu mate 16.04 LTS[14] | 無し |
| ソフトウェア | ROS2 Ardent Apalone[2] | FreeRTPS[7] |

表 2　ROS2 および軽量 DDS のデフォルトの QoS 設定

| QoS | ROS2（publisher） | ROS2（subscriber） | FreeRTPS |
|---|---|---|---|
| History | Keep last | Keep last | Keep last |
| Depth | 7 | 5 | 1 |
| Reliability | Reliable | Best effort | Best effort |
| Durability | Volatile | Volatile | 設定不可 |

**（項目 1.4）QoS の設定による通信時間の変化**

ROS2 および軽量 DDS の QoS の設定を変更して通信時間を測定した．ROS2 は Reliability と Durability を，軽量 DDS は Reliability を変更して計測した．

**（項目 2）ROS2 および軽量 DDS の初期化に要する時間**

ROS2 および軽量 DDS の初期化に要する時間を計測するために，アプリケーションが起動して main 関数が呼び出された直後から subscriber が最初にメッセージを受信してコールバック関数を呼び出すまでを計測し，これを初期化に要する時間とした．また，初期化に要する時間はノードが publisher と subscriber の場合の両方を計測した．

**（項目 3）軽量 DDS のメモリ使用量**

軽量 DDS のメモリ使用量を計測するために，”objdump -h [オブジェクトファイル]”コマンドを使用して，低性能 CPU に書き込むオブジェクトファイルのセクションヘッダ情報を調べた．メモリ使用量としては，ROM と RAM の使用量を計測した．

### 5.2　評価結果

**（項目 1）ROS2 および軽量 DDS の通信時間**

まず，図 4 と表 3 に各組み合わせで通信をした際の通信時間を計測した結果を示す．縦軸は通信時間を表しており，通信時間の分布を示している．

**（項目 1.1）時間経過による通信時間の変化**

時間経過による通信時間の変化を図 5 に示す．また，図 5 に示している通信時間はそれぞれ，100 秒間の平均を示している．

**（項目 1.2）メッセージサイズによる通信時間の変化**

まず，表 4 に各通信の送信したメッセージのサイズと，そのメッセージを送信した際の，パケットのサイズを示す．メッセージサイズよる通信時間の変化を図 6 に示す．また，図 6 に示している通信時間はそれぞれ，100 秒間の平均を示している．

**（項目 1.3）ノード数による通信時間の変化**

ノード数による通信時間の変化を計測した結果を表 5 に示す．増やしたノードは publisher か subscriber のどちらのノード数を増やしたかを示している．ノード数はノードいくつまで増やしたかを示している．ノード 1 からノード 4 までは各ノードの通信時間の平均と最悪値を示している．

**（項目 1.4）QoS の設定による通信時間の変化**

ROS2 の通信時間を計測した際の QoS の組み合わせを表 6 に示す．CPU 内での ROS2 の通信時間を計測した際の結果を図 7a に，CPU 間での ROS2 の通信時間を図 7b に示す．それぞれの凡例は QoS の組み合わせを示しており，表 6 に対応している．また，軽量 DDS の通信時間を表 7c に示す．各計測結果は平均と最悪値を示している．

**（項目 2）ROS2 および軽量 DDS の初期化に要する時間**

ROS2 と軽量 DDS の初期化に要する時間を図 8 に示す．軽量 DDS の publisher が ROS2 と通信する際は通信を開始することができず，初期化に要する時間を計測することができなかった．

**（項目 3）軽量 DDS のメモリ使用量**

軽量 DDS の ROM および RAM の使用量を表 7 に示す．

### 5.3　考察

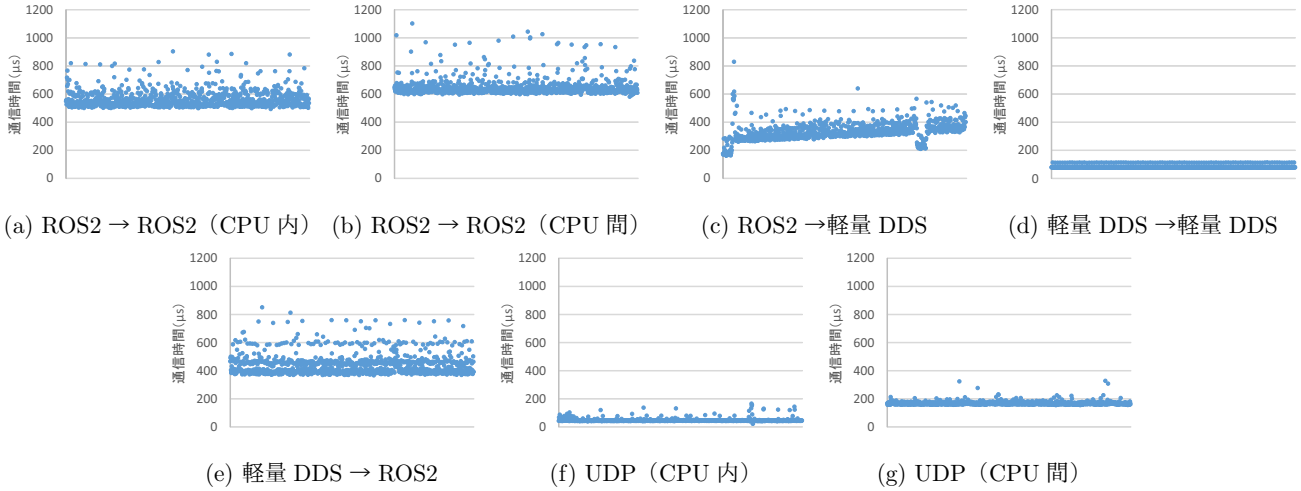
**（項目 1）ROS2 および軽量 DDS の通信時間**

表 3 より，ROS2 は軽量 DDS に比べて通信時間が大きいことがわかった．また，通信時間のばらつきも軽量 DDS に比べて大きいことがわかった．UDP の通信時間は ROS2 の通信時間と比較して小さいため，ROS2 の通信時間は通信以外の処理に起因するものが大きいと考えられる．これは，ROS2 が汎用 OS である Linux 上で動作していることが原因だと考えられる．よって，現状の ROS2 を自律制御システムに適用することは現実的ではないと考えられる．

一方，軽量 DDS は ROS2 に比べて通信時間が小さいことがわかった．また，通信時間のばらつきも小さいことがわかった．これは，軽量 DDS が DDS の機能を完全に備えていないため，行う処理が少ないことが原因だと考えられる．リアルタイム性の面では，軽量 DDS を自律制御システムに適用することは現実的だと考えられる．

**（項目 1.1）時間経過による通信時間の変化**

図 5 より，ROS2 から軽量 DDS にメッセージを送信す




(a) ROS2 → ROS2（CPU 内） (b) ROS2 → ROS2（CPU 間） (c) ROS2 →軽量 DDS (d) 軽量 DDS →軽量 DDS

(e) 軽量 DDS → ROS2 (f) UDP（CPU 内） (g) UDP（CPU 間）

**図 4** 各組み合わせの通信時間

**表 3** ROS2，軽量 DDS，UDP の通信時間

| | 通信回数 | 平均（$\mu$s） | 最悪値（$\mu$s） | 最良値（$\mu$s） | 標準偏差（$\mu$s） |
|---|---|---|---|---|---|
| ROS2 → ROS2（CPU 内） | 998 | 562 | 904 | 494 | 59.4 |
| ROS2 → ROS2（CPU 間） | 998 | 647 | 1102 | 581 | 61.1 |
| ROS2 →軽量 DDS | 997 | 332 | 830 | 162 | 63.7 |
| 軽量 DDS →軽量 DDS | 1000 | 82.7 | 114 | 78 | 9.94 |
| 軽量 DDS → ROS2 | 1000 | 441 | 851 | 367 | 75.5 |
| UDP（CPU 内） | 999 | 46.8 | 166 | 23 | 12.4 |
| UDP（CPU 間） | 999 | 168 | 328 | 155 | 13.1 |

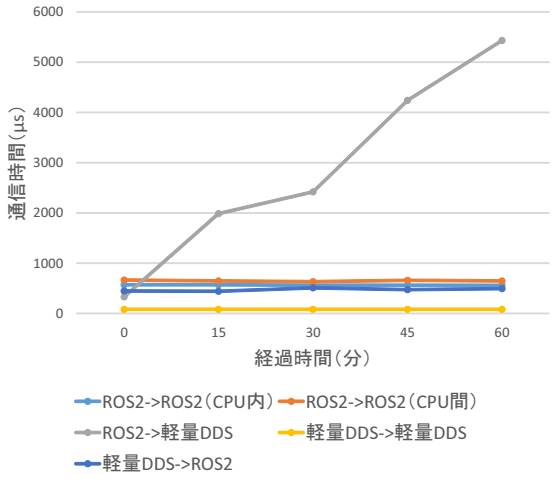



**図 5** 時間経過による通信時間の変化

**表 4** 各送信元のメッセージサイズとパケットサイズ

| | メッセージサイズ（byte） | | | | |
|---|---|---|---|---|---|
| 送信元 | 64 | 128 | 256 | 512 | 1024 |
| ROS2 | 222 | 286 | 414 | 670 | 1182 |
| 軽量 DDS | 171 | 235 | 363 | 619 | 1126 |
| UDP | 106 | 170 | 298 | 554 | 1066 |

る際に，時間が経過するにつれて，通信時間が大きくなることがわかった．しかし，通信時間が大きくなる原因はわかっておらず，今後，原因を調査する必要がある．

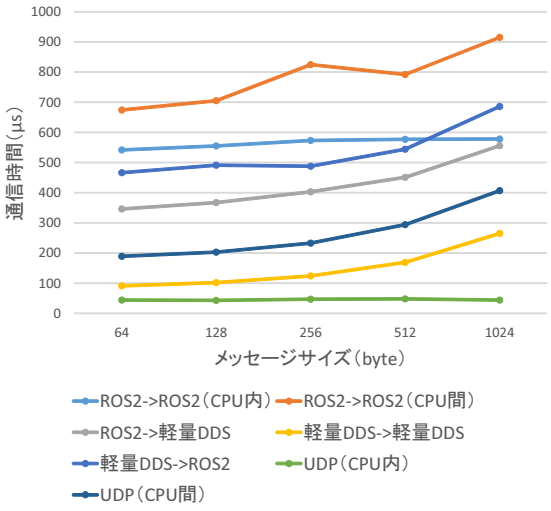


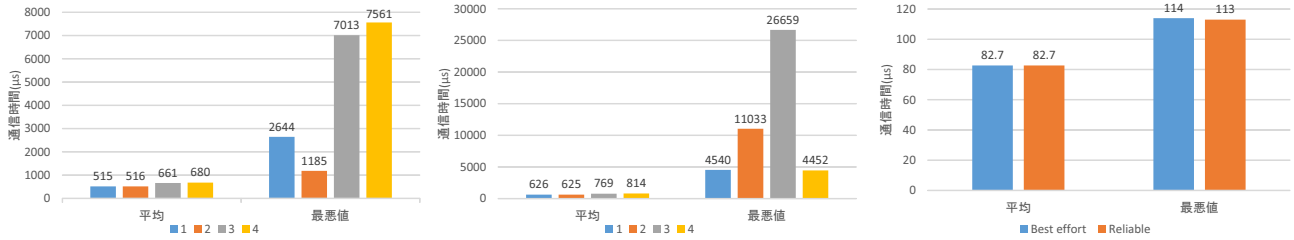

**図 6** メッセージサイズによる通信時間の変化

**（項目 1.2）メッセージサイズによる通信時間の変化**

図 6 より，マイコン内で行われる通信はメッセージサイズが大きくなっても，通信時間が変化しないことがわかった．一方，マイコン間で行われる通信はメッセージサイズが大きくなるにつれて，通信時間が大きくなることがわかった．マイコン間で通信を行う際には LAN を介して通信を行うため，LAN を介して通信を行う際にメッセージサイズの影響を受けると考えられる．また，マイコン間の

表 5　ノード数による通信時間の変化（平均/最悪値（$\mu$s））

| 増やしたノード | ノード数 | ノード 1 | ノード 2 | ノード 3 | ノード 4 |
|---|---|---|---|---|---|
| publisher | 2 | 984/712 | 563/13756 | - | - |
| publisher | 3 | 763/7138 | 1149/6119 | 957/7480 | - |
| publisher | 4 | 606/2643 | 965/6423 | 710/3770 | 1077/4728 |
| subscriber | 2 | 364/586 | 364/586 | - | - |
| subscriber | 3 | 391/639 | 391/639 | 391/638 | - |
| subscriber | 4 | 408/1074 | 408/1074 | 408/1074 | 407/1073 |




(a) ROS2 → ROS2（CPU 内）　(b) ROS2 → ROS2（CPU 間）　(c) 軽量 DDS →軽量 DDS

図 7　QoS の設定による通信時間の変化

表 6　QoS の組み合わせ

| QoS ポリシ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| Reliability | Best effort | Best effort | Reliable | Reliable |
| Durability | Volatile | Transient local | Volatile | Transient local |

表 7　軽量 DDS のメモリ使用量

| | ROM | RAM |
|---|---|---|
| publisher | 58,672byte | 107,136byte |
| subscriber | 58,660byte | 107,136byte |

ROS2 の通信時間と UDP の通信時間を比較すると，通信時間の変化の割合が類似しているため，メッセージサイズは通信以外の処理のオーバヘッドに影響を与えないと考えられる．文献 [10] によると，PC 上で ROS2 を評価した場合も，PC 内の通信はメッセージサイズによる通信時間の変化が小さく，PC 間の通信ではメッセージサイズによる通信時間の変化が大きいという結果が得られているため，妥当な結果だと考えられる．

**（項目 1.3）ノード数による通信時間の変化**

表 5 より，publisher のノード数を増やした場合と比較して，subscriber のノードを増やした場合のほうが，通信時間のばらつきが小さいことが分かった．これは，ROS2 が複数の軽量 DDS にメッセージを送信する際に，ブロードキャスト通信を行っており，軽量 DDS が同じメッセージを受け取るために，通信時間が近い値になることが原因だと考えられる．

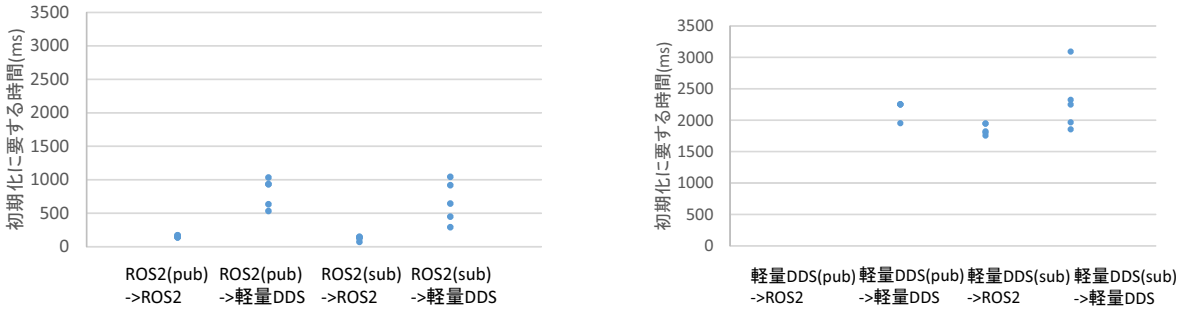
**（項目 1.4）QoS の設定による通信時間の変化**

図 7a および図 7b より，ROS2 は QoS の設定により通信時間が変化することがわかった．QoS ポリシについて，Reliability が Best effort の場合の方が通信時間が小さいことがわかった．また，Durability が Volatile の場合の方が通信時間が小さいことがわかった．Best effort はメッセージの送信を失敗したときに再送をしない設定のため，再送する設定である Reliable と比較して，送信の失敗を確認する処理を必要としない分オーバヘッドは小さいと考えられる．Volatile は古いメッセージを保存しない設定のため，保存する設定である Transient local と比較して，保存する処理を行わない分オーバヘッドは小さいと考えられる．そのため，結果は妥当なものだと考えられる．最悪値については，平均値の 30 倍以上大きい値の結果が出ている組み合わせもある．QoS の組み合わせと関連するのか，Linux のスケジューリング等と関連するのか等の調査が必要であるが今後の課題とする．

一方，図 7c より，軽量 DDS は QoS の設定を変更しても通信時間が変わらないことがわかった．これは QoS が通信時間に影響を与えないのか，影響が観測できない程小さいかはわからず，さらに調査をする必要がある．

**（項目 2）ROS2 および軽量 DDS の初期化に要する時間**

図 8 より，ROS2 と軽量 DDS 共に初期化に要する時間は動作周波数と比較すると明らかに長いことがわかり，自律制御システムで頻繁に ECU の電源の ON と OFF が切り替わる場合には問題があると考えられる．また，ROS2 と軽量 DDS を比較すると，ROS2 の方が初期化に要する時間が短いことがわかった．これはマイコンの性能の差と，通信相手を探すために周期的に行われる処理の周期の違いが原因だと考えられる．メッセージの送信周期によっても初期化に要する時間は変化すると考えられるため，送信周期を変更して計測を行う必要がある．さらに，軽量 DDS の publisher が ROS2 と通信する場合においては通信を開始することができず，原因もわかっていないため，今後原因を調査する必要がある．




(a) ROS2　　(b) 軽量 DDS

**図 8**　初期化に要する時間

**（項目 3）軽量 DDS のメモリ使用量**

本研究で使用した低性能 CPU は 1Mbyte のフラッシュメモリと 192Kbyte の RAM を持っている．図 7 より，ROM は容量の約 6%を使用している．また，RAM は容量の約 55%を使用している．よって，ROM は容量に十分な余裕があることがわかった．一方，RAM はバッファやスタックにメモリを使用している可能性があるため，今後調査する必要がある．

## 6. おわりに

本研究では，サービスロボットや自動運転の自律制御システムに ROS2 および軽量 DDS を使用することを想定して，適用性の評価を行った．自律制御システムでは，高性能 CPU では通信に ROS2 を，低性能 CPU では通信に軽量 DDS を使用することを想定して，評価環境と評価項目を設定した．評価実験を行った結果，ROS2 は特に通信以外にかかるオーバヘッドが大きく，自律制御システムに適用するには課題があることがわかった．一方，軽量 DDS は通信時間が小さく，自律制御システムへの適用が望める結果となった．また，ROS2 や軽量 DDS の初期化には長い時間がかかることがわかり，軽量 DDS のメモリ使用量は評価実験で使用したマイコンでは RAM を約 55%の容量を使用していることがわかった．

今後の課題としては，本研究で行った評価実験の結果の中で原因のわからなかった点について調査することがあげられる．

### 参考文献

[1] Open Source Robotics Foundation: Documentation - ROS Wiki, Open Source Robotics Foundation (online), available from ⟨http://wiki.ros.org/⟩ (accessed 2018-2-8).
[2] Open Source Robotics Foundation: Home ros2/ros2 Wiki GitHub, Open Source Robotics Foundation (online), available from ⟨http://github.com/ros2/ros2/wiki⟩ (accessed 2018-2-8).
[3] Object Management Group: DDS Portal - Data Distribution Services, Object Management Group (online), available from ⟨http://portals.omg.org/dds/⟩ (accessed 2018-2-8).
[4] Raspberry Pi Foundation: Raspberry Pi 3 Model B - Raspberry Pi, Raspberry Pi Foundation (online), available from ⟨https://www.raspberrypi.org/products/raspberry-pi-3-model-b/⟩ (accessed 2018-2-8).
[5] STMicroelectronics: *Discovery kit with STM32F407VG MCU* (2017).
[6] eProsima: eProsima Fast RTPS, eProsima (online), available from ⟨http://www.eprosima.com/index.php/products-all/eprosima-fast-rtps⟩ (accessed 2018-2-8).
[7] Open Source Robotics Foundation: GitHub - ros2/freertps: a free, portable, minimalist, work-in-progress RTPS implementation, Open Source Robotics Foundation (online), available from ⟨https://github.com/ros2/freertps⟩ (accessed 2018-2-8).
[8] Open Source Robotics Foundation: DDS and ROS middleware implementations ros2/ros2 Wiki GitHub, Open Source Robotics Foundation (online), available from ⟨https://github.com/ros2/ros2/wiki/DDS-and-ROS-middleware-implementations⟩ (accessed 2018-2-8).
[9] Open Source Robotics Foundation: About Quality of Service Settings ros2/ros2 Wiki GitHub, Open Source Robotics Foundation (online), available from ⟨https://github.com/ros2/ros2/wiki/About-Quality-of-Service-Settings⟩ (accessed 2018-2-8).
[10] Maruyama, Y., Kato, S. and Azumi, T.: Exploring the Performance of ROS2, *Proceedings of the 13th International Conference on Embedded Software*, EMSOFT '16, New York, NY, USA, ACM, pp. 5:1–5:10 (online), DOI: 10.1145/2968478.2968502 (2016).
[11] Tardioli, D., Parasuraman, R. and Ögren, P.: Pound: A ROS node for Reducing Delay and Jitter in Wireless Multi-Robot Networks, *CoRR*, Vol. abs/1707.07540 (online), available from ⟨http://arxiv.org/abs/1707.07540⟩ (2017).
[12] 森田錬，松原克弥：ROS2 における通信特性に応じた DDS 実装の動的選択機構の実現，第 47 回組込みシステム合同研究発表会 (ETNET2018)，島根県隠岐郡 (Mar 2018).
[13] Embest Technology Co., LTD: *STM32F4DIS-BB User Manual* (2012).
[14] Ubuntu MATE Team: Ubuntu MATE — For a retrospective future, Ubuntu MATE Team (online), available from ⟨https://ubuntu-mate.org/⟩ (accessed 2018-2-8).